



City Research Online

City, University of London Institutional Repository

Citation: Filho, Gilberto Amado de Azevedo Cysneiros (2011). Software Traceability for Multi-Agent Systems Implemented Using BDI Architecture. (Unpublished Doctoral thesis, City University London)

This is the unspecified version of the paper.

This version of the publication may differ from the final published version.

Permanent repository link: <https://openaccess.city.ac.uk/id/eprint/1115/>

Link to published version:

Copyright: City Research Online aims to make research outputs of City, University of London available to a wider audience. Copyright and Moral Rights remain with the author(s) and/or copyright holders. URLs from City Research Online may be freely distributed and linked to.

Reuse: Copies of full items can be used for personal research or study, educational, or not-for-profit purposes without prior permission or charge. Provided that the authors, title and full bibliographic details are credited, a hyperlink and/or URL is given for the original metadata page and the content is not changed in any way.

City Research Online:

<http://openaccess.city.ac.uk/>

publications@city.ac.uk

Software Traceability for Multi-Agent Systems implemented using BDI Architecture

Gilberto A. de A. Cysneiros Filho

A thesis submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy at City University London

City University London
Department of Computing

June 2011

Volume 2

Contents

Volume 1

Contents	2
Figures.....	7
Tables.....	9
Acknowledgements.....	10
Declaration.....	11
Abstract.....	12
Chapter 1 - Introduction.....	13
1.2 Hypotheses.....	20
1.3 Objectives	20
1.4 Contributions.....	21
1.5 Thesis Outline	22
Chapter 2 - Literature Survey on Traceability	24
2.1 Traceability Reference Models and Meta-Models.....	27
2.2 Traceability Approaches to Capture Trace Relations	33
2.2.1 Formal Approaches.....	34
2.2.2 Process Oriented Approaches	34
2.2.3 Information Retrieval Approaches.....	36
2.2.4 String Matching Approaches	39
2.2.5 Rule Based Approaches	40
2.2.6 Run-time approaches	43
2.2.7 Hypermedia and Information Integration approaches	44
2.3 Representation, Recording and Maintenance of Traceability Relations.....	45
2.4 Visualisation of Traceability Relations.....	46
2.5 Use of Traceability Relations.....	47
2.6 Traceability Approaches for Multi-Agent Systems	49
2.7 Performance Measures.....	50
2.8 Implication of tools that infer trace relations.....	51
2.9 Summary	52
Chapter 3 - Traceability Reference Model.....	54
3.1 Overview of the Reference Model.....	54
3.2 Multi-agent Oriented Artefacts.....	56
3.2.1 i* Framework.....	56
3.2.2 Prometheus.....	60
3.2.3 JACK.....	67
3.3 Traceability Relations	73
3.3.1 Traceability Relations between i* and Prometheus	74
3.2.2 Traceability Relations between Prometheus and JACK.....	86
3.4 Summary	95
Chapter 4 - Traceability Framework.....	96
4.1. Overview of the Framework	96
4.2 Traceability and Completeness Checking Rules.....	101
Type 1:	104

Type 2:	112
Type 3:	114
4.3 Extended Functions.....	116
4.3.1 Completeness checking functions.....	117
4.3.2 XQuery functions.....	119
4.3.3 XQueryJACKFunctions.....	121
4.3.4 XQueryPDTFunctions	122
4.3.5 XQuerySimilarityFunctions.....	123
4.3.6 XQuerySynonymsFunctions.....	125
4.3.7 XQueryTAOMFunctions	128
4.4 Retratos Tool.....	129
4.5 Discussion	139
4.6 Summary	140
Chapter 5 - Evaluation and Results.....	141
5.1 Criteria for Evaluation	142
5.2 Automatic Teller Machine	143
5.2.1 Overview of the Case Study	143
5.2.2 Artefacts.....	145
5.2.3 Evaluation	146
5.3 Air Traffic Control Environment.....	155
5.3.1 Overview of the Case Study	155
5.3.2 Artefacts.....	156
5.3.3 Evaluation	157
5.4 Electronic Bookstore.....	164
5.4.1 Overview of the Case Study	164
5.4.2 Artefacts.....	165
5.4.3 Evaluation	167
5.5 Discussion.....	168
5.6 Threats of Validity	170
5.7 Summary	172
Chapter 6 - Conclusion and Future Works	173
6.1 Overall Conclusions.....	173
6.2 Hypotheses.....	178
6.3 Objectives	181
6.4 Contributions.....	182
6.5 Future Work.....	183
6.5 Final Remarks	186
Bibliography	187

Volume 2

Contents	2
Figures.....	7
Tables.....	12
Appendix A - Extended Functions.....	15
A.1.1 Completeness checking functions.....	16
A.1.2 XQuery functions.....	22

A.1.3 XQueryJACKFunctions	30
A.1.4 XQueryPDTFunctions	32
A.1.4.1 ActorHasCapability function	34
A.1.4.2 FieldTokenizer function.....	35
A.1.4.3 GetAttributeValue function.....	36
A.1.4.4 GetIncludedFields function.....	37
A.1.4.5 GetInformationCarried function	38
A.1.4.6 GetPDTFileName.....	38
A.1.4.7 GetPrometheusElements	39
A.1.4.8 GetPrometheusSubElements	40
A.1.4.9 GetPrometheusStepScenarios	41
A.1.4.10 GetPrometheusSubGoalsElements.....	42
A.1.4.11 GetPrometheusSubGoalElements	43
A.1.4.12 GetPrometheusUsesData.....	44
A.1.4.13 IsACapabilityThatTheAgentIncludes	44
A.1.4.14 IsADataProducedByTheRole.....	44
A.1.4.15 IsADataThatTheAgentReads	44
A.1.4.16 IsADataThatTheAgentWrites	44
A.1.4.17 IsADataThatTheCapabilityReads	45
A.1.4.18 IsADataThatTheCapabilityWrites	45
A.1.4.19 IsADataThatThePlanReads.....	45
A.1.4.20 IsADataThatThePlanWrites.....	45
A.1.4.21 IsADataUsedByTheRole.....	45
A.1.4.22 IsADataThatTheAgentAchieves	45
A.1.4.23 IsAGoalThatTheCapabilityAchieves	46
A.1.4.24 IsAGoalThatThePlanAchieves.....	46
A.1.4.25 IsAGoalThatTheAgentAchieves.....	46
A.1.4.26 IsAGoalThatTheCapabilityAchieves	46
A.1.4.27 IsAMessageThatTheAgentReceives	46
A.1.4.28 IsAMessageThatTheAgentSends	47
A.1.4.29 IsAMessageThatTheCapabilityReceives	47
A.1.4.30 IsAMessageThatTheCapabilitySends	47
A.1.4.31 IsAMessageThatTheReceives.....	47
A.1.4.32 IsAMessageThatThePlanReceives.....	47
A.1.4.33 IsAMessageThatThePlanSends.....	48
A.1.4.34 IsAMessageThatTriggersThePlan.....	48
A.1.4.35 IsAnActionThatTheAgentPerforms	48
A.1.4.36 IsAnActionThatTheCapabilityPerforms	48
A.1.4.37 IsAnActionThatThePlanPerforms.....	48
A.1.4.38 IsAPerceptThatTheAgentResponds	49
A.1.4.39 IsAPerceptThatTheCapabilityResponds	49
A.1.4.40 IsAPerceptThatThePlanResponds.....	49
A.1.4.41 IsAPerceptThatTheCapabilityResponds	49
A.1.4.42 IsAPerceptThatThePlanResponds.....	49
A.1.4.43 IsAPerceptThatTheCapabilityResponds	50
A.1.4.44 IsAPerceptThatThePlanResponds.....	50

A.1.4.45 IsAPlanThatTheAgentIncludes	50
A.1.4.46 IsAPlanThatTheCapabilityIncludes	50
A.1.4.47 IsAPlanTheRoleUses	51
A.1.4.48 IsARoleThatTheAgentIncludes	51
A.1.4.49 IsTrigger.....	51
A.1.5 XQuerySimilarityFunctions	51
A.1.6 XQuerySynonymsFunctions	64
A.1.7 XQueryTAOMFunctions	69
Appendix B – Automated Teller Machine.....	73
B.1 Introduction	73
B.2 Organizational Models	74
B.3 Prometheus Models	76
B.4 JACK Code	78
B.5 JACK Code in XML	86
B.6 Evaluation.....	95
Appendix C – Air Traffic Control Environment.....	105
C.1 Introduction	105
C.2 Organizational Models	106
C.3 Prometheus Models	107
C.4 JACK Code	113
C.5 Code in XML	128
C.6 Evaluation.....	140
Appendix D – Electronic Bookstore Case Study	155
D.1 JACK Agent vs Prometheus Goal.....	155
D.2 JACK Agent vs Prometheus Role.....	157
D.3 JACK Agent vs Prometheus Agent.....	158
D.4 JACK Agent vs Prometheus Capability.....	159
D.5 JACK Agent vs Prometheus Plan	160
D.6 JACK Agent vs Prometheus Percept	162
D.7 JACK Agent vs Prometheus Action.....	164
D.8 JACK Agent vs Prometheus Message (sends).....	165
D.9 JACK Agent vs Prometheus Message (receives).....	166
D.10 JACK Agent vs Prometheus Data (uses)	167
D.11 JACK Agent vs Prometheus Data (creates)	168
D.12 JACK Plan vs Prometheus Goal	168
D.13 JACK Plan vs Prometheus Role.....	169
D.14 JACK Plan vs Prometheus Agent	170
D.15 JACK Plan vs Prometheus Capability	172
D.16 JACK Plan vs Prometheus Plan	174
D.17 JACK Plan vs Prometheus Percept	178
D.18 JACK Plan vs Prometheus Action (Sends)	179
D.19 JACK Plan vs Prometheus Message (Sends).....	180
D.20 JACK Plan vs Prometheus Message (Receives)	182
D.21 JACK Plan vs Prometheus Data (Uses)	184
D.22 JACK Plan vs Prometheus Data (Creates).....	185
D.23 JACK BeliefSet vs Prometheus Role (Creates)	185

D.24 JACK BeliefSet vs Prometheus Role (Uses)	185
D.25 JACK BeliefSet vs Prometheus Role (Creates)	186
D.26 JACK BeliefSet vs Prometheus Agent (Uses)	187
D.27 JACK BeliefSet vs Prometheus Capability (Creates)	187
D.28. JACK BeliefSet vs Prometheus Capability (Uses)	188
D.29 JACK BeliefSet vs Prometheus Plan (Creates).....	189
D.30 JACK BeliefSet vs Prometheus Plan (Uses).....	189
D.31 JACK BeliefSet vs Prometheus Data.....	190
D.32 JACK Event vs Prometheus Agent (sends).....	191
D.33 JACK Event vs Prometheus Agent (receives)	192
D.34 JACK Event vs Prometheus Capability (sends).....	193
D.35. JACK Event vs Prometheus Capability (receives)	193
D.36 JACK Event vs Prometheus Plan (sends)	194
D.37 JACK Event vs Prometheus Plan (receives)	196
D.38 JACK Event vs Prometheus Message.....	197
Appendix E – Introduction to BDI architecture.....	199
E.1 Agent Architectures.....	199
E.2 BDI Architecture	199
Appendix F - Traceability Relations between i^* and Prometheus.....	202
Appendix G - Traceability Relations between Prometheus and JACK	243

Figures

Figure A.1 Calling <i>getPDFFileName</i> extended function in Java	16
Figure A.2 List of strings.....	19
Figure A.3 <i>getDocSourceMissingElement</i> function.....	20
Figure A.4 <i>getIDMissingElement</i> function example.....	21
Figure A.5 <i>getNameMissingElement</i> function example.....	21
Figure A.6 <i>getNumberOfMissingElement</i> function example	22
Figure A.7 Arrival Sequencing Capability and ATL SD Resource	24
Figure A.8 <i>capabilityUsesSDResource</i> function example	25
Figure A.9 <i>hasUses</i> function example	26
Figure A.10 <i>contains</i> function example	27
Figure A.11 Using <i>contains</i> function.....	28
Figure A.12 <i>getAttributeValue</i> function example	28
Figure A.13 <i>hasRelation</i> function example	29
Figure A.14 <i>stringTokenizer</i> function example.....	30
Figure 4.15 <i>stringTokenizerByUpperCase</i> function example.....	30
Figure A.16 <i>getBeliefSetFields</i> function example.....	31
Figure A.17 <i>ActorHasCapability</i> function example	35
Figure A.18 <i>fieldTokenizer</i> function example.....	36
Figure A.19 <i>getAttributeValue</i> function example	36
Figure A.20 <i>getIncludesFields</i> function example	37
Figure A.21 <i>getInformationCarried</i> function example	38
Figure A.22 <i>getPrometheusElements</i> function example	39
Figure A.23 <i>getPrometheusSubElements</i> function example	40
Figure A.24 <i>getPrometheusStepScenarios</i> function example	41
Figure A.25 <i>getPrometheusSubGoals</i> function example	42
Figure A.26 <i>getPrometheusSubGoalsElements</i> function example.....	43
Figure A.27 <i>hasUses</i> function example	53
Figure A.28 <i>hasUses</i> function example	54
Figure A.29 <i>creates</i> function example.....	54
Figure A.30 <i>creates</i> function example.....	55
Figure A.31 <i>overlaps</i> function example.....	56
Figure A.32 <i>overlaps</i> function example.....	57
Figure A.33 <i>isPositiveSimilar</i> function example	58
Figure A.34 <i>isSimilar</i> function example	59
Figure A.35 <i>isSimilarByOverlaps</i> function example	60
Figure A.36 <i>isSimilarDataAndBeliefSet</i> function example.....	61
Figure A.37 <i>isSimilarSDResourceAndMessage</i> function example	62
Figure A.38 <i>SomeOverlap</i> function example.....	63
Figure A.39 <i>stringTokenizerByUpperCase</i> function example	64
Figure A.40 <i>contains</i> function example.....	65
Figure A.41 <i>isSynonyms</i> function	66
Figure A.42 <i>stringTokenizer</i> function example.....	67
Figure A.43 <i>isSynonyms</i> function example	68
Figure A.44 <i>getSubGoalsAndTask</i> function example	70
Figure A.45 <i>getSubGoalsAndTask</i> function example	71
Figure A.46 <i>getAttributeValue</i> function example	72
Figure B.1 Strategic Dependency model for the Automatic Teller Machine.....	75
Figure B.2 Strategic Rationale Model for the Automatic Teller Machine.....	76
Figure B.3 ATM Goal diagram	76
Figure B.4 ATM System Overview diagram.....	77
Figure B.5 – Atm Agent Overview Diagram	78
Figure B.6 Bank Agent Overview diagram.....	78
Figure B.7 Atm agent	80

Figure B.8 BankAgent agent	80
Figure B.9 Accounts beliefSet	81
Figure B.10 Accounts beliefSet	82
Figure B.11 Withdraw event.....	83
Figure B.12 WithdrawResponse event.....	83
Figure B.13 WithdrawRequest event.....	84
Figure B.14 WithdrawApproved plan.....	84
Figure B.15 WithdrawCash plan.....	85
Figure B.16 WithdrawRejected plan	85
Figure B.17 WithdrawApproved plan.....	86
Figure B.18 Atm agent in XML	88
Figure B.19 BankAgent in XML	88
Figure B.20 Accounts beliefSet in XML	90
Figure B.21 Balances beliefSet in XML	91
Figure B.22 ProcessWithdraw plan in XML	92
Figure B.23 WithdrawApproved plan in XML	92
Figure B.24 WithdrawCash plan in XML.....	93
Figure B.25 WithdrawRejected plan in XML	93
Figure B.26 Withdraw event in XML.....	94
Figure B.27 WithdrawResponse in XML.....	94
Figure B.29 Fields of the Accounts beliefSet.....	99
Figure B.30 Accounts beliefSet	99
Figure B.31 Balances beliefSet	99
Figure B.32 Balances descriptor.....	100
Figure B.33 ProcessWithdraw plan	101
Figure B.34 Process Withdraw descriptor	101
Figure B.35 WithdrawApproved plan.....	102
Figure B.36 Withdraw Approved descriptor	102
Figure B.37 WithdrawCash plan.....	103
Figure B.38 Withdraw Cash descriptor.....	103
Figure B.39 WithdrawReject plan	103
Figure B.40 Withdraw Rejected descriptor.....	104
Figure C.1 Strategic Dependency model for Air Traffic Environment.....	106
Figure C.2 Strategic Rationale model for Air Traffic Environment.....	107
Figure C.3 Goal diagram for Air Traffic Environment	108
Figure C.4 Traffic Feeding Capability.....	109
Figure C.5 Arrival Sequencing Capability.....	109
Figure C.6 Runway Assigning Capability	110
Figure C.7 Flying Capability	110
Figure C.8 Traffic Feeding Capability.....	111
Figure C.9 Arrival Sequencing Capability.....	111
Figure C.10 Runway Assigning Capability	112
Figure C.11 Flying Capability	113
Figure C.12 Aircraft agent.....	113
Figure C.13 Airport agent.....	114
Figure C.14 Feeder agent	114
Figure C.15 Runway agent.....	114
Figure C.16 LandingInfo beliefSet	115
Figure C.17 RunwayInfo beliefSet.....	116
Figure C.18 ArrivalSequencing capability.....	116
Figure C.19 Flying Capability	117
Figure C.20 Runway Assigning Capability	117
Figure C.21 TrafficFeeding capability.....	118
Figure C.22 AircraftEvent event	119
Figure C.23 Approaching event	120
Figure C.24 EnterControlArea.....	120

Figure C.25 TrafficEvent event	121
Figure C.26 AssignSlot plan.....	121
Figure C.27 FollowApproach plan	122
Figure C.28 InitialApproach plan.....	122
Figure C.29 MonitorAircraft plan.....	123
Figure C.30 RequestSlot plan.....	124
Figure C.31 RunwayAssign plan.....	125
Figure C.32 RunwayRequest plan.....	126
Figure C.33 Takeoff plan	127
Figure C.34 TakeoffDiscard plan.....	127
Figure C.35 Traffic plan.....	128
Figure C.36 Aircraft agent in XML.....	128
Figure C.37 Airport agent in XML.....	129
Figure C.38 TrafficFeeding agent in XML.....	129
Figure C.39 Runway agent in XML.....	129
Figure C.40 LandingInfo beliefSet in XML.....	130
Figure C.41 RunwayInfo beliefSet in XML.....	131
Figure C.42 AssignSlot plan in XML.....	131
Figure C.43 FollowApproach plan in XML	131
Figure C.44 InitialApproach plan in XML.....	132
Figure C.45 MonitorAircraft plan in XML.....	132
Figure C.46 RequestSlot plan in XML.....	134
Figure C.47 RunwayAssign plan in XML.....	134
Figure C.48 Takeoff plan in XML	135
Figure C.49 TakeoffDiscard plan in XML.....	135
Figure C.50 Traffic plan in XML.....	135
Figure C.51 RunwayRequest plan in XML.....	136
Figure C.52 AircraftEvent event in XML	136
Figure C.53 Approaching event in XML	137
Figure C.54 EnterControlArea event in XML.....	137
Figure C.55 TrafficEvent event in XML.....	138
Figure C.56 ArrivalSequencing capability in XML.....	138
Figure C.57 Flying capability in XML	139
Figure C.58 RunwayAssigning capability in XML.....	139
Figure C.59 TrafficFeeding capability in XML.....	140
Figure C.60 Air Traffic Control Environment i* model version 1	147
Figure C.61 Prometheu goal diagram.....	147
Figure E.1 A generic BDI architecture	201
Figure F.1 Prometheus Goal vs SD Goal overlaps dependency	206
Figure F.2 Monitor Shipment task dependency.....	207
Figure F.3 Monitor delivery goal in Prometheus.....	207
Figure F.4 Prometheus Goal vs Actor depends on traceability relation.....	208
Figure F.5 Prometheus Goal vs SR Goal overlaps traceability relation.....	209
Figure F.6 Prometheus Goal vs SR Task overlaps traceability relation	210
Figure F.7 Prometheus Role vs SD Goal uses traceability relation.....	210
Figure F.8 Prometheus Role vs SD Resources uses relation.....	211
Figure F.9 Prometheus Role vs SD Task contributes relation	212
Figure F.10 Prometheus Role vs Actor contributes relation.....	213
Figure F.11 Prometheus Role vs SR Goal achieves traceability relation	213
Figure F.12 Prometheus Role vs SR Resource uses relation.....	214
Figure F.13 Prometheus Role vs SR Resource creates relation.....	215
Figure F.14 Prometheus Role vs SR Task achieves traceability relation.....	215
Figure F.15 Prometheus Agent vs SD Goal achieves traceability relation.....	216
Figure F.16 Prometheus Agent vs SR Resource uses traceability relation	216
Figure F.17 Prometheus Agent vs SD Task achieves traceability relation	217
Figure F.18 Prometheus Agent vs Istar Actor overlaps traceability relation	217

Figure F.19 Prometheus Agent vs SR Goal achieves traceability relation	218
Figure F.20 Prometheus Agent vs SR Resource uses traceability relation	219
Figure F.21 Prometheus Agent vs SR Resource creates traceability relation.....	219
Figure F.22 Prometheus Agent vs SR Task achieves traceability relation.....	220
Figure F.23 Prometheus Capability vs SD Goal contributes traceability relation	221
Figure F.24 Prometheus Capability vs SD Resource uses traceability relation	221
Figure F.25 Prometheus Capability vs SD Task contributes traceability relation.....	222
Figure F.26 Prometheus Capability vs Actor composed relation	223
Figure F.27 Prometheus Capability vs SR Goal contributes traceability relation.....	223
Figure F.28 Prometheus Capability vs SR Resource uses traceability relation.....	224
Figure F.29 Prometheus Capability vs SR Resource creates traceability relation	224
Figure F.30 Prometheus Capability vs SR Resource uses traceability relation.....	225
Figure F.31 Prometheus Plan vs SD Goal contributes traceability relation	225
Figure F.32 Prometheus Plan vs SD Resource uses traceability relation.....	226
Figure F.33 Prometheus Plan vs SD Task achieves traceability relation.....	227
Figure F.34 Prometheus vs Actor creates traceability relation	228
Figure F.35 Prometheus Plan vs SR Goal achieves traceability relation	228
Figure F.36 Prometheus Plan vs SR Resource uses traceability relation	229
Figure F.37 Prometheus Plan vs SR Resource creates traceability relation	229
Figure F.38 Prometheus Plan vs SR Task achieves traceability relation.....	230
Figure F.39 Prometheus Percept vs SD Resource overlaps traceability relation.....	230
Figure F.40 Prometheus Action vs SR Task overlaps traceability relation.....	231
Figure F.41 Prometheus Data vs SD Goal contributes traceability relation	232
Figure F.42 Prometheus Data vs SD Task contributes traceability relation	233
Figure F.43 Prometheus Data vs Actor uses traceability relation	234
Figure F.44 Prometheus Data vs SR Goal uses traceability relation	235
Figure F.45 Prometheus Data vs SR Resource overlaps traceability relation.....	235
Figure F.46 Prometheus Data vs SR Task uses traceability relation.....	236
Figure F.47 Order Book Scenario.....	237
Figure F.48 Strategic Rationale Diagram for the Electronic Bookstore actor	238
Figure F.49 Prometheus Scenario vs SD Task depends traceability relation	239
Figure F.50 Prometheus Scenario vs Actor depends traceability relation.....	239
Figure F.51 Prometheus Scenario vs SR Goal compose traceability relation	240
Figure F.52 Prometheus Scenario vs SR Resource creates traceability relation.....	241
Figure F.53 Prometheus Scenario vs SR Resource uses traceability relation	241
Figure F.54 Prometheus Scenario vs SR Task composed traceability relation.....	242
Figure F.55 Prometheus Message vs SD Resource overlaps traceability relation	242
Figure G.1 JACK Agent vs Prometheus Goal achieves traceability relation.....	245
Figure G.2 JACK Agent vs Prometheus Role uses traceability relation	246
Figure G.3 JACK Agent vs Prometheus Agent overlaps traceability relation	246
Figure G.4 JACK Agent vs Prometheus Capability uses traceability relation.....	247
Figure G.5 JACK Agent vs Prometheus Plan uses traceability relation	247
Figure G.6 JACK Agent vs Prometheus Percept uses traceability relation	248
Figure G.7 JACK Agent vs Prometheus Action creates traceability relation.....	249
Figure G.8 JACK Agent vs Prometheus Message sends traceability relation.....	249
Figure G.9 JACK Agent vs Prometheus Message receives traceability relation	250
Figure G.10 JACK Agent vs Prometheus Message receives traceability relation	250
Figure G.11 JACK Agent vs Prometheus Date creates traceability relation	251
Figure G.12 JACK Plan vs Prometheus Goal	251
Figure G.13 JACK Plan vs Prometheus Role uses traceability relation	252
Figure G.14 JACK Plan vs Prometheus Agent uses traceability relation	253
Figure G.15 JACK Plan vs Prometheus Capability uses traceability relation.....	253
Figure G.16 JACK Plan vs Prometheus Plan overlaps traceability relation.....	254
Figure G.17 JACK Plan vs Prometheus Percept uses traceability relation.....	254
Figure G.18 JACK Plan vs Prometheus Action creates traceability relation	255
Figure G.19 JACK Plan vs Prometheus Message sends traceability relation	255

<i>Figure G.20 JACK Plan vs Prometheus Message receives traceability relation</i>	256
<i>Figure G.21 JACK Plan vs Prometheus Data uses traceability relation</i>	256
<i>Figure G.22 JACK Plan vs Prometheus Data creates traceability relation</i>	257
<i>Figure G.23 JACK Belief vs Prometheus Role creates relation</i>	257
<i>Figure G.24 JACK Belief vs Prometheus Role uses traceability relation</i>	258
<i>Figure G.25 JACK BeliefSet vs Prometheus Agent creates traceability relation</i>	258
<i>Figure G.26 JACK BeliefSet vs Prometheus Agent uses traceability relation</i>	259
<i>Figure G.27 JACK BeliefSet vs Prometheus Capability creates traceability relation</i>	260
<i>Figure G.28 JACK BeliefSet vs Prometheus Capability uses traceability relation</i>	260
<i>Figure G.29 JACK BeliefSet vs Prometheus Plan creates traceability relation</i>	261
<i>Figure G.30 JACK BeliefSet vs Prometheus uses traceability relation</i>	262
<i>Figure G.31 JACK BeliefSet vs Prometheus Data overlaps traceability relation</i>	262
<i>Figure G.32 JACK Event vs Prometheus Agent receives traceability relation</i>	263
<i>Figure G.33 JACK Event vs Prometheus Agent sends traceability relation</i>	263
<i>Figure G.34 JACK Event vs Prometheus Capability</i>	264
<i>Figure G.35 JACK Event vs Prometheus Capability receives relation</i>	264
<i>Figure G.36 JACK Event vs Prometheus Plan sends traceability relation</i>	265
<i>Figure G.37 JACK Event vs Prometheus Plan receives traceability relation</i>	266
<i>Figure G.38 JACK Event vs Prometheus Message overlaps traceability relation</i>	266

Tables

<i>Table A.1- Completeness checking functions</i>	17
<i>Table A.2 XQuery functions</i>	23
<i>Table A.3 XQueryJACKFunctions</i>	31
<i>Table A.4 XQueryPDTFunctions</i>	34
<i>Table A.5 XQuerySimilarityFunctions</i>	52
<i>Table A.6 XQuerySynonyms Function example</i>	64
<i>Table A.7 XQueryTAOMFunctions</i>	69
<i>Table B.1 Traceability relations identified manually</i>	96
<i>Table B.2 Traceability relations identified by the tool</i>	98
<i>Table B.3 Missing Information</i>	98
<i>Table C.1 Traceability relations identified manually</i>	142
<i>Table C.2 Traceability relations identified by the tool</i>	144
<i>Table C.3 Missing relations identified by the tool</i>	145
<i>Table C.4 Missing relations between SD Goal and Prometheus Goal</i>	146
<i>Table C.5 Missing relations between SR Goal and Prometheus Goal</i>	146
<i>Table C.6 Missing relations between SR Plan and Prometheus Goal</i>	147
<i>Table C.7 Missing relations between Prometheus Goal and SD/SR Task or SD/SR Goal</i>	148
<i>Table C.8 Missing relations between SD Resource and Prometheus Percept</i>	148
<i>Table C.9 Missing relations between SD Goal and a Prometheus Agent</i>	149
<i>Table C.10 Missing relation between a SR Goal and an Agent</i>	149
<i>Table C.11 Missing relations between a SR Task and an Agent</i>	149
<i>Table C.12 Missing relations between a SD Goal and a Prometheus Plan</i>	150
<i>Table C.13 Missing relations between a SR Goal and a Prometheus Plan</i>	150
<i>Table C.14 Missing relations between a SR Task and Prometheus Plan</i>	150
<i>Table C.15 Missing links between a SD Goal and Prometheus Capability</i>	151
<i>Table C.16 Missing links between a SR Goal and Prometheus Capability</i>	151
<i>Table C.17 Missing relations between a SR Task and Prometheus Capability</i>	151
<i>Table C.18 Traceability relations between i^* and Prometheus</i>	154
<i>Table C.19 Missing relations between JACK and Prometheus</i>	154
<i>Table D.1 Relations identified manually between Prometheus Goal and JACK Agent</i>	156
<i>Table D.2 Relations identified by the tool between Prometheus Goal and JACK Agent</i>	157
<i>Table D.3 Relations identified manually between Prometheus Role and JACK Agent</i>	157
<i>Table D.4 Relations identified by the tool between Prometheus Role and JACK Agent</i>	158
<i>Table D.5 Relations identified manually between Prometheus Agent and JACK Agent</i>	158
<i>Table D.6 Relations identified by the tool between Prometheus Agent and JACK Agent</i>	158
<i>Table D.7 Relations identified manually Prometheus Capability and JACK Agent</i>	159
<i>Table D.8 Relations identified by the tool between Prometheus Capability and JACK Agent</i>	160
<i>Table D.9 Relations identified manually between Prometheus Plan and JACK Agent</i>	161
<i>Table D.10 Relations identified by the tool between Prometheus Plan and JACK Agent</i>	162
<i>Table D.11 Relations identified manually between Prometheus Percept and JACK Agent</i>	163
<i>Table D.12 Relations identified by the tool between Prometheus Percept and JACK Agent</i>	163
<i>Table D.13 Missing traceability relations between Prometheus Percept and JACK Agent</i>	164
<i>Table D.14 Relations identified manually between Prometheus Action and JACK Agent</i>	164
<i>Table D.15 Relations identified by the tool between Prometheus Action and JACK Agent</i>	165
<i>Table D.16 Relations identified manually between Prometheus Message and JACK Agent</i>	166
<i>Table D.17 Relations identified by the tool between Prometheus Message and JACK Agent</i>	166
<i>Table D.18 Relations identified manually between Prometheus Message and JACK Agent</i>	167
<i>Table D.19 Relations identified by the tool between Prometheus Message and JACK Agent</i>	167
<i>Table D.20 Relations identified manually between Prometheus Data and JACK Agent</i>	168
<i>Table D.21 Relations identified by the tool between Prometheus Data and JACK Agent</i>	168
<i>Table D.22 Relations identified manually between Prometheus Data and JACK Agent</i>	168
<i>Table D.23 Relations identified by the tool between Prometheus Data and JACK Agent</i>	168

Table D.24 Relations identified manually between JACK Plan and Prometheus Goal	169
Table D.25 Relations identified by the tool between Prometheus Goal and JACK Plan	169
Table D.26 Relations identified manually between JACK Plan and Prometheus Role	170
Table D.27 Relations identified by the tool between Prometheus Role and JACK Plan	170
Table D.28 Relations identified manually between JACK Plan and Prometheus Agent	171
Table D.29 Relations identified by the tool between Prometheus Agent and JACK Plan	172
Table D.30 Relations identified manually between JACK Plan and Prometheus Capability.....	173
Table D.31 Relations identified by the tool between Prometheus Capability and JACK Plan.....	174
Table D.32 Relations identified manually between Prometheus Plan and JACK Plan	175
Table D.33 Relations identified by the tool between Prometheus Plan and JACK Plan.....	175
Table D.34 Missing relation.....	176
Table D.35 Missing relation.....	176
Table D.36 Wrong relation.....	176
Table D.37 Missing relation.....	177
Table D.38 Wrong relation.....	177
Table D.39 Missing relations	177
Table D.40 Missing relation.....	178
Table D.41 Relations not identified by the tool	178
Table D.42 Relations identified manually between JACK Plan and Prometheus Percept.....	179
Table D.43 Relations identified by the tool between Prometheus Percept and JACK Plan.....	179
Table D.44 Relations identified manually between JACK Plan and Prometheus Action.....	180
Table D.45 Relations identified by the tool between Prometheus Action and JACK Plan.....	180
Table D.46 Relations identified manually between JACK Plan and Prometheus Message	181
Table D.47 Relations identified by the tool between Prometheus Message and JACK Plan.....	181
Table D.48 Relations identified manually between JACK Plan and Prometheus Message	183
Table D.49 Relations identified by the tool between Prometheus Message and JACK Plan.....	183
Table D.50 Relations identified manually between JACK Plan and Prometheus Data	184
Table D.51 Relations identified by tool between JACK Plan and Prometheus Data	184
Table D.52 Relations identified manually between JACK Plan and Prometheus Data	185
Table D.53 Relations identified manually between JACK BeliefSet and Prometheus Role	185
Table D.54 Relations identified by the tool between Prometheus Role and JACK BeliefSet.....	185
Table D.55 Relations identified manually between JACK BeliefSet and Prometheus Role	186
Table D.56 – Relations identified by the tool between JACK BeliefSet and Prometheus Role.....	186
Table D.57 Relations identified manually between JACK BeliefSet and Prometheus Agent	186
Table D.58 Relations identified by the tool between Prometheus Role	186
Table D.59 Relations identified manually between JACK BeliefSet and Prometheus Agent	187
Table D.60 Relations identified by the tool between Prometheus Agent and JACK BeliefSet.....	187
Table D.61 Relations identified manually between Prometheus Capability and JACK BeliefSet.....	188
Table D.62 Relations identified by the tool between Prometheus Capability and JACK BeliefSet.....	188
Table D.63 Relations identified manually between Prometheus Capability and JACK BeliefSet.....	188
Table D.64 Relations identified by the tool between Prometheus Capability and JACK BeliefSet.....	189
Table D.65 Relations identified manually between Prometheus Plan and JACK	189
Table D.66 Relations identified by the tool between Prometheus Plan and JACK BeliefSet	189
Table D.67 Relations identified manually between Prometheus Plan and JACK BeliefSet	190
Table D.68 Relations identified by the tool between Prometheus Plan and JACK BeliefSet	190
Table D.69 Relations between JACK BeliefSet and Prometheus Data.....	191
Table D.70 Relations between Prometheus Data and JACK BeliefSet.....	191
Table D.71 Relations between JACK Event and Prometheus Agent	192
Table D.72 Relations between JACK Event and Prometheus Agent	192
Table D.73 Relations between JACK Event and Prometheus Capability.....	193
Table D.74 Relations identified manually between JACK Event and Prometheus Capability.....	194
Table D.75 Relations identified manually between JACK Event and Prometheus Plan	195
Table D.76 Relations identified by the tool between Prometheus Message and Prometheus Plan	196
Table D.77 Relations identified manually between JACK Event and Prometheus Plan	197
Table D.78 Relations identified by the tool between Prometheus Plan and JACK Event	197
Table D.79 Relations between JACK Event and Prometheus Message.....	198

<i>Table D.80 Relations between Prometheus Message and JACK Event.....</i>	<i>198</i>
<i>Table F.1 Relations between Prometheus and i* SD.....</i>	<i>204</i>
<i>Table F.2 Relations between Prometheus and i*SR elements</i>	<i>205</i>
<i>Table G.1 Traceability Relations Types between Prometheus and JACK Artefacts.....</i>	<i>244</i>
<i>Table G.2 Traceability Relations Types between Prometheus and JACK Artefacts.....</i>	<i>244</i>

Appendix A - Extended Functions

The lists of functions are grouped in seven classes:

- XQueryCompletenessCheckingFunctions - contains a list of methods in Java that extend XQuery to perform completeness checking.
- XQueryFunctions – contains a list of methods in Java that extend XQuery with general functionalities.
- XQueryJACKFunctions - provides a list of methods in Java that that extends XQuery with functions to manipulate elements in the JACK XML file.
- XQueryPDTFunctions - provides a list of methods in Java that extends XQuery with functions to manipulate elements created by the PDT tool version 3.2.
- XQuerySimilarityFunctions – includes a list of methods in Java that extends XQuery with functions to compare the similarity between elements in the models.
- XQuerySynonymsFunctions – contains a list of methods in Java that extends XQuery with functions to verify if names of elements in the models are synonyms.
- XQueryTAOMFunctions – provides a list of methods in Java that extends XQuery with functions to manipulate elements in the *i** model created using the TAOM tool.

To use an XQuery extended function implemented in a Java class, it is necessary first to declare the class name that includes the function and then call the function wanted. For instance, before to be able to call *getPDTFileName* function included in the XQueryPDTFunctions class the user has first to define a namespace and associated it with XQueryPDTFunctions class and then call the function wanted using the namespace given. Figure A.1 shows an example when the pdt namespace is associated to the XQueryPDTFunction class in Java (*declare namespace pdt = java:retratos.XQueryPDTFunctions*). The *getPDTFileName* function is invoked using the namespace followed by colon and the function name (*pdt:getPDTFileName()*).

```
declare namespace pdt = "java:retratos.XQueryPDTFunctions"; ....  
let $pdtDoc := doc(pdt:getPDTFileName())
```

Figure A.1 Calling getPDTFileName extended function in Java

The next sections describe functions implemented in Java to extend XQuery in more detail.

A.1.1 Completeness checking functions

Table A.1 shows a list of methods in the XQueryCompletenessCheckingFunctions class that extend XQuery with functions to perform completeness checking. We are going to describe in detail and give some examples of the most complex and important functions in next sections.

Method Summary	
boolean	<u>completenessChecking</u> (ArrayList< <u>TraceElement</u> > list1, ArrayList< <u>TraceElement</u> > list2) Verify if list2 contains elements with the same (or synonyms) name of elements in the list1 Elements in list1 that it does not have an element in list2 with the same name (or synonyms) are added to ArrayList missingElements
double	<u>getDegreeOfCompleteness</u> () Returns degree of similarity between two elements.
String	<u>getDocSourceMissingElement</u> (int i) Returns document file name from the source element that is missing to be represented in the target document.
String	<u>getDocSourceMissingElementA</u> (int i) Returns name of the element that is missing to be represented in the target document.
String	<u>getIDMissingElement</u> (int i) Returns id of the missing element.
String	<u>getIDMissingElementA</u> (int i) Returns id from source element that is missing to be represented in the target document.
String	<u>getNameMissingElement</u> (int i) Returns name of the missing element.
String	<u>getNameMissingElementA</u> (int i) Returns name of the element that is missing to be represented in the target document.
int	<u>getNumberOfElements</u> () Returns number of an element that is missing to be represented in the target document.
int	<u>getNumberOfElements</u> (ArrayList< <u>TraceElement</u> > elements) Returns the size of the ArrayList elements
int	<u>getNumberOfMissingElements</u> () Returns the size of ArrayList missingElement minus one
int	<u>getNumberOfMissingElementsA</u> () Returns the size of the ArrayList elements
String	<u>getTypeOfMissingElement</u> (int i) Returns type of the missing element.
String	<u>getTypeOfMissingElementA</u> (int i) Returns the name of the type from source element that is missing to be represented in the target document.
String	<u>getTypeTargetMissingElement</u> (int i) Returns document file name from the target element that is missing to be represented

Table A.1- Completeness checking functions

A.1.1.1 CompletenessChecking functions

The completenessChecking function can be used to verify the similarity between two lists of elements. The function receives two lists of elements and compares if names of the elements are synonyms. For instance, in the Figure A.2, we have the ListA that contains “Login outgoing delivery”, “Calculate delivery time estimates”, and “Get delivery options” and the ListB that contains “Obtain Delivery Options”, “Compute Delivery Estimates” and “Place Delivery Request”. The function completenessChecking checks if each element of the ListA has a synonym in the ListB. In the Figure A.5, “Get delivery options” is a synonym to “Obtain Delivery Options”, and “Calculate delivery time estimates” is a synonym to “Compute Delivery Time Estimates”. The “Login outgoing delivery” does not have synonyms in the List B. The “Login outgoing delivery” element is added to a list of missing elements.

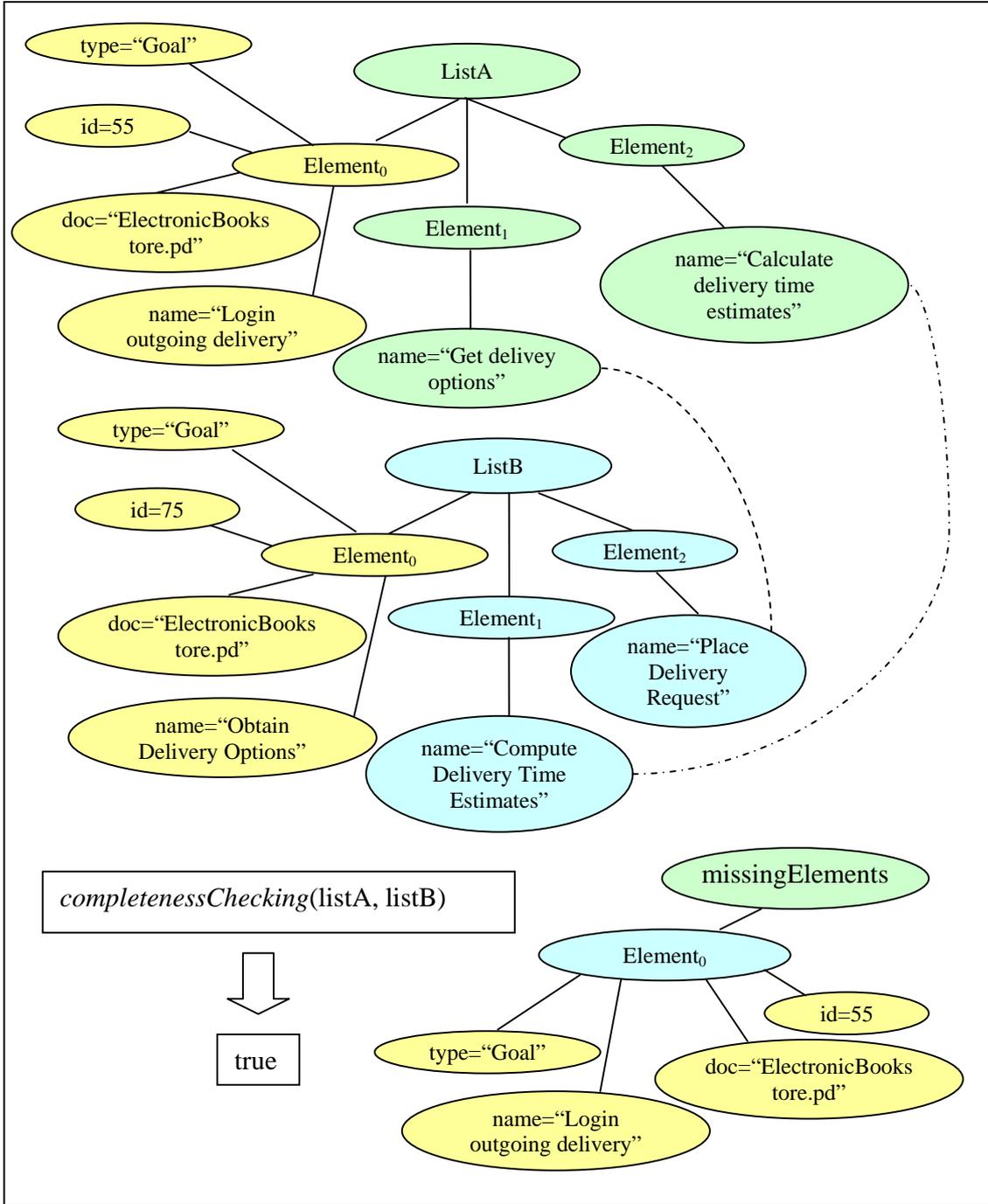


Figure A.2 List of strings

A.1.1.2 GetDegreeOfCompleteness function

The getDegreeOfCompleteness function returns the value of degreeOfCompleteness field.

A.1.1.3 GetDocSourceMissingElement

The `getDocSourceMissingElement` function receives as parameter an integer that represents the index of the missing element and returns a String with the name of the document that contains the element that is missing. For instance, in the Figure A.3 the `Element0` (index equal 0) is part of the `ElectronicBookstore.pd` document. If the `getDocSourceMissingElement` function is called passing the value equal 0 then `getDocSourceMissingElement` function returns the “`ElectronicBookstore.pd`” string value.

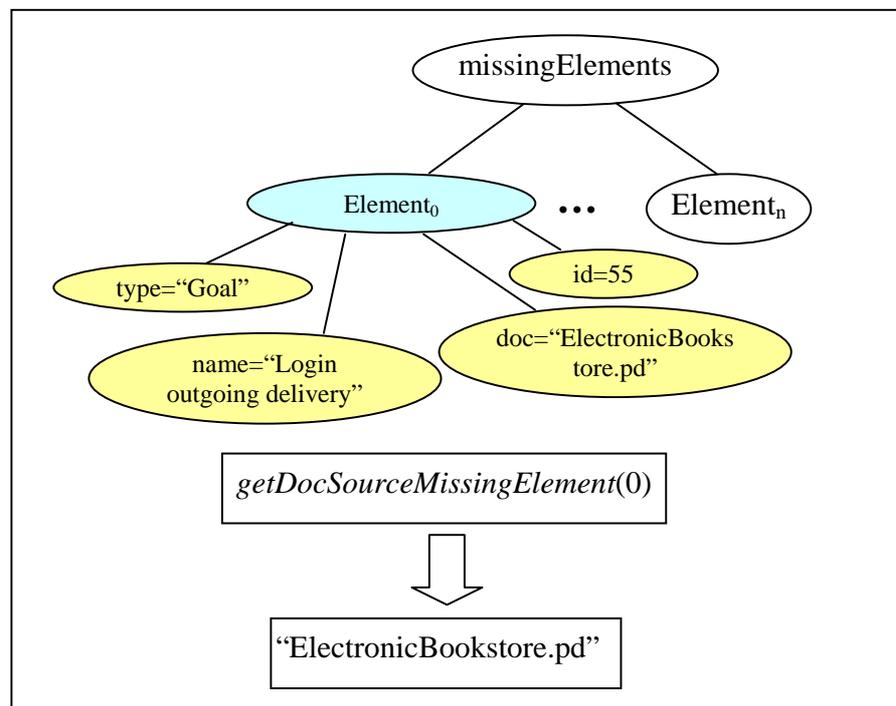


Figure A.3 `getDocSourceMissingElement` function

A.1.1.4 GetIDMissingElement

The `getIDMissingElement` function receives as parameter an integer that represents the index of the missing element and returns a String with the id of the element that is missing. For instance, in the Figure A.4 the `Element0` (index equal 0) has id equal to 0. If the `getIDMissingElement` function is called passing the value equal 0 then function returns the value 55.

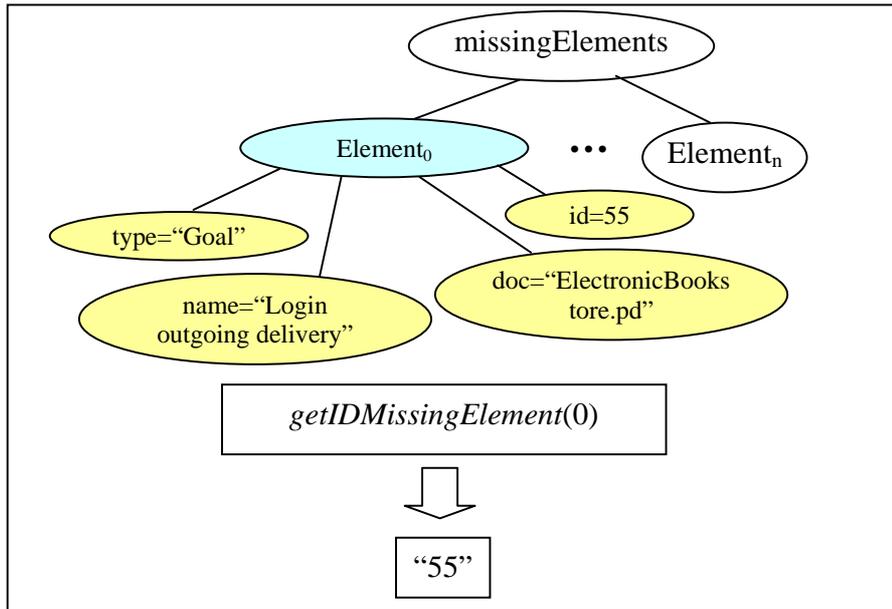


Figure A.4 `getIDMissingElement` function example

A.1.1.5 `GetNameMissingElement`

The `getNameMissingElement` function receives as parameter an integer that represents the index of the missing element and returns a String with the name of the element that is missing. For instance, in the Figure A.5 if the `getNameMissingElement` function is called passing the value equal 0 then function returns the “Login outgoing delivery” string value.

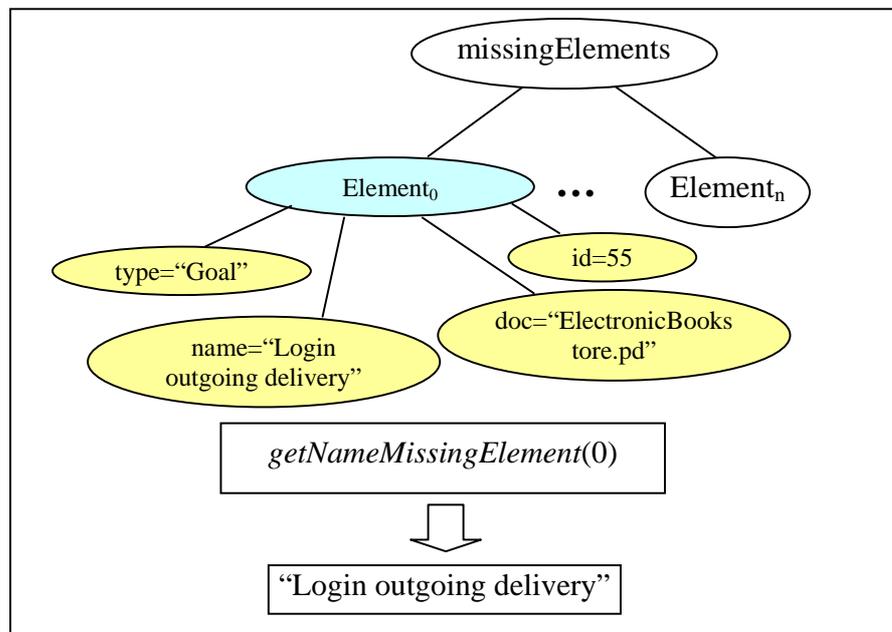


Figure A.5 `getNameMissingElement` function example

A.1.1.6 GetNumberOfMissingElement

The *getNumberOfMissingElement* function returns the size of ArrayList *missingElement* minus one. For instance, in the Figure A.6 the *getNumberOfMissingElement* function returns the value *n-1*.

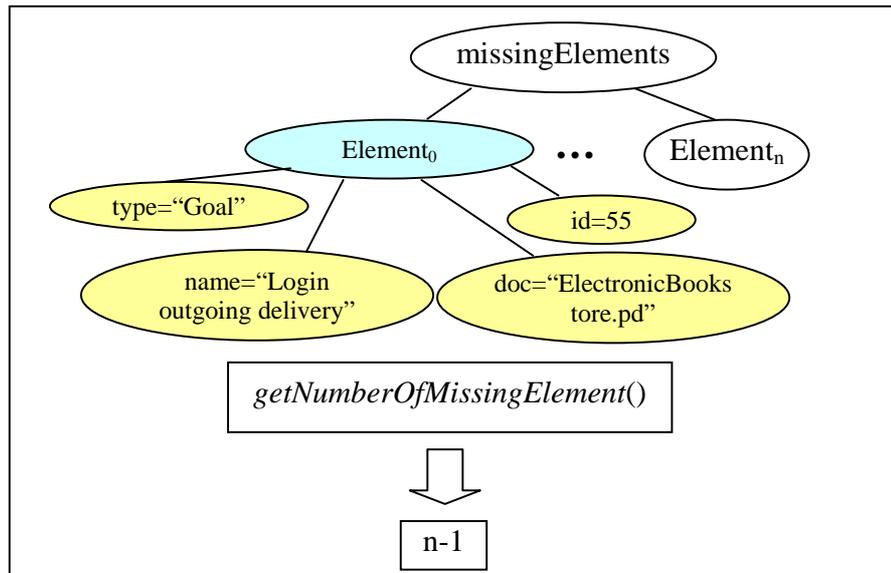


Figure A.6 *getNumberOfMissingElement* function example

A.1.1.7 GetDocSourceMissingElementA

```
return XQueryFunctions.sourceMissingElements.get(i).getDoc();
```

A.1.2 XQuery functions

Method Summary	
boolean	capabilityUsesSDResource (TinyNodeImpl capability, TinyNodeImpl resource) Check if the Capability in Prometheus uses a SD Resource in <i>i</i> *
boolean	capabilityUsesSRResource (TinyNodeImpl capability, TinyNodeImpl resource) Check if the Capability in Prometheus uses a SR Resource in <i>i</i> *
boolean	clr () Reset value of class variables used to perform completeness

	checking: numberOfElements , missingElements, numberOfMissingElements and sourceMissingElements
boolean	<u>contains</u> (ArrayList<String> list1, ArrayList<String> list2) Check if the ArrayList of Strings list1 contains all the strings in the ArrayList of Strings list2
boolean	<u>contains</u> (String word, ArrayList<String> wordList) Check if the ArrayList of String wordList contains the String word.
String	<u>getAttributeValue</u> (TinyNodeImpl node, attributeName) Returns the value of an attribute of an XML Element in Saxon The method compares if any of the messages that the Capability sends or receives has an overlaps relations with the SR Resource.
String	<u>getTraceabilityFileName</u> () It returns a String of tokens.
boolean	<u>hasRelation</u> (elementID, String relationType, String type) Check if there is a specific type of relations between an element in which the id is equal to elementID and any element with the same type as type
ArrayList<String>	<u>stringTokenizer</u> (String str) It returns a String of tokens.
ArrayList<String>	<u>stringTokenizerByUpperCase</u> (String str) It returns a String of tokens.

Table A.2 XQuery functions

A.1.2.1 CapabilityUsesSDResource function

The `capabilityUsesSDResource` function checks if a capability uses a SD Resource. The function receives as a parameter `TinyNodeImpl` capability and `TinyNodeImpl` resource that represents a XML Node in the Saxon tool. For instance, the Figure A.7 shows the Arrival Sequencing Capability and the ATL SD Resource. The Arrival Sequential Capability contains the Aircraft Event message. If you call the `capabilityUsesSDResource` function (see Figure A.8) and pass as argument the Arrival Sequencing Capability `TinyNodeImpl` and the ATL `TinyNodeImpl`, the function recovers all messages that the capability contains and then checks if there is some overlaps relation between the id of a message and the id of the SD resource using the `isOverlap` function.

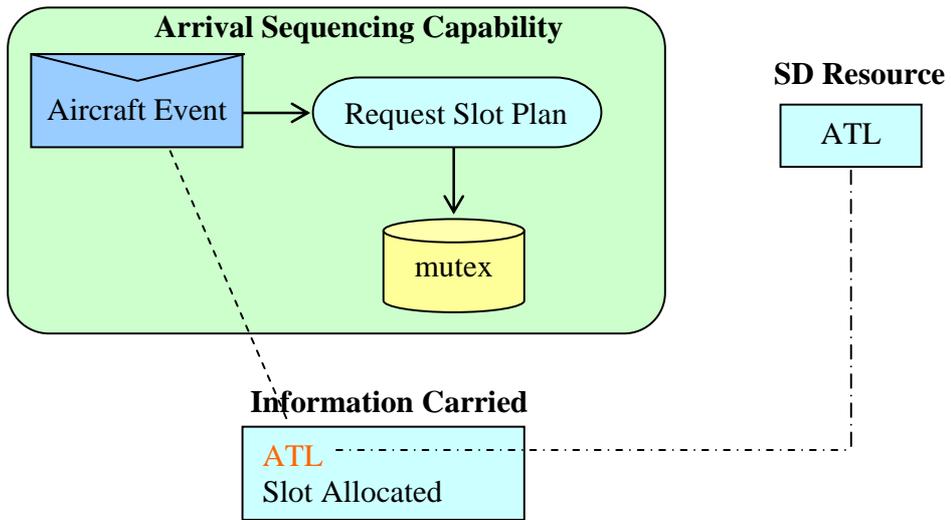


Figure A.7 Arrival Sequencing Capability and ATL SD Resource

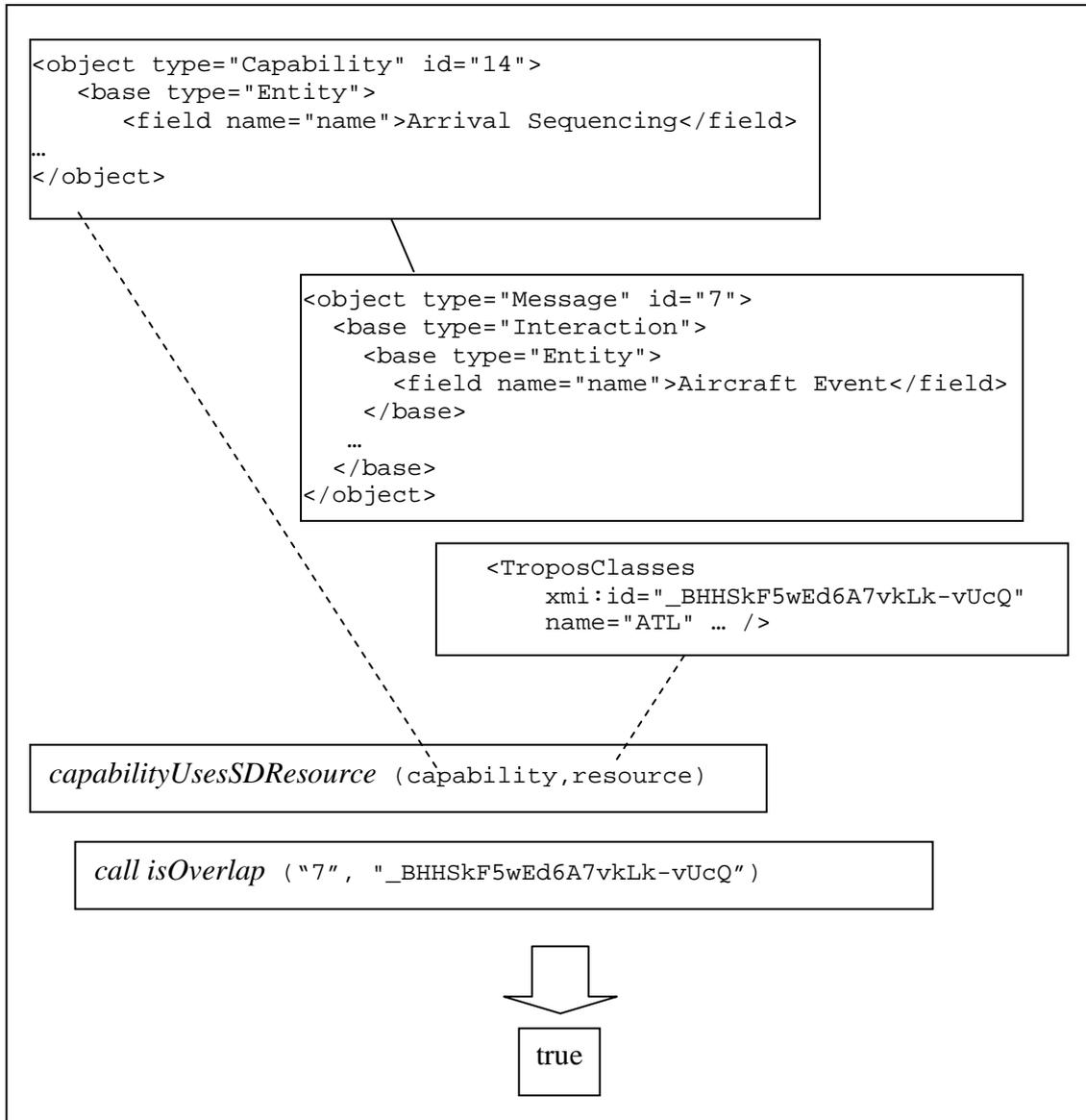


Figure A.8 *capabilityUsesSDResource* function example

A.1.2.2 *CapabilityUsesSRResource* function

The *capabilityUsesSRResource* function checks if a capability uses a SR Resource. The function receives as a parameter *TinyNodeImpl* capability and *TinyNodeImpl* resource that represents a XML Node in the Saxon tool. For instance, the Figure A.9 shows the Flying Capability and the Landing Information SR Resource. If you call the

capabilityUsesSRResource function and pass as argument the Flying Capability TinyNodeImpl and the LandingInformation TinyNodeImpl.

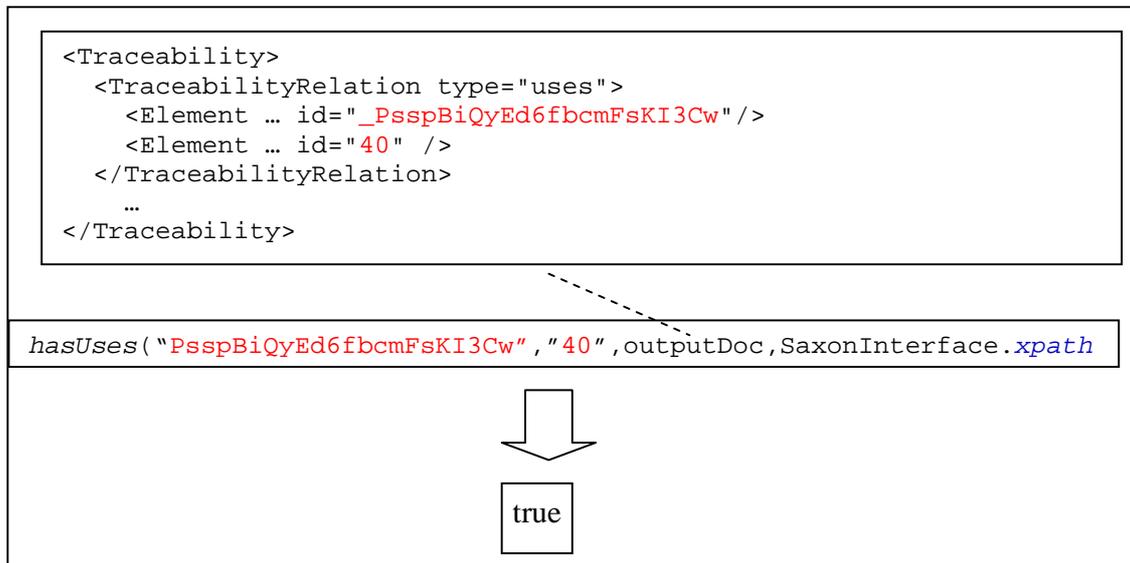


Figure A.9 hasUses function example

A.1.2.3 Contains function

Contains function receives as parameter a String word and an ArrayList of String wordList and check if the ArrayList wordList contains the String word. The function uses the WordNet dictionary to check for synonyms words. Figure A.10 shows that the contains function returns true when it is invoked passing as parameter the String “Get” and the list of Strings wordList that consists of “Obtain”, “Delivery” and “Options”. The list of Strings wordList contains the “Get” string because “Get” and “Obtain” are synonyms.

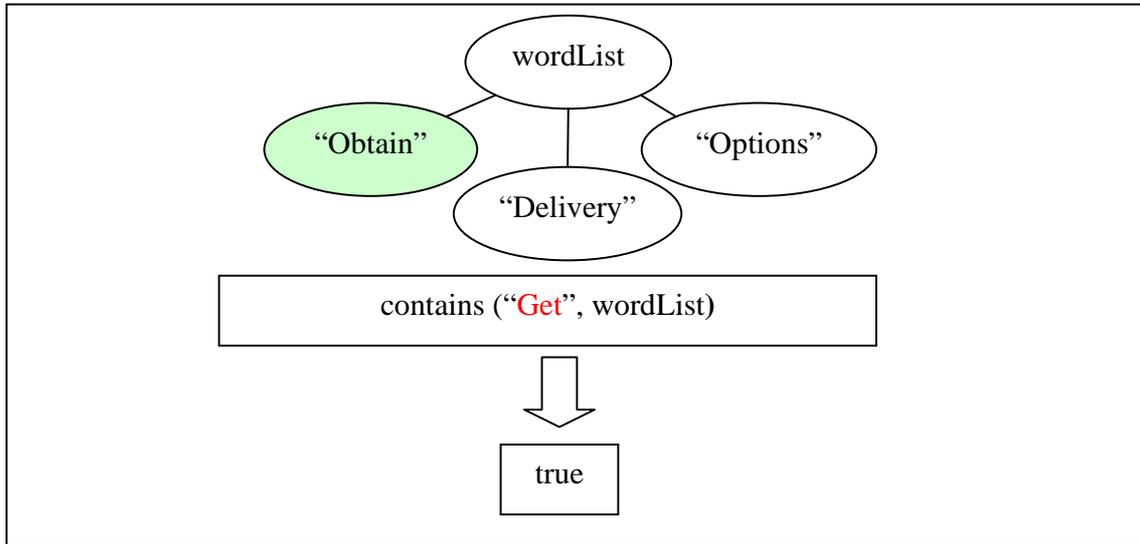


Figure A.10 contains function example

A.1.2.4 Contains function

Contains function check if the ArrayList of Strings List A contains all the strings in the ArrayList of Strings List B. To each String in the List A the contains function call the other contains function explained in the section A.1.2.3 passing the String in the List A and the List B as parameter. If all the Strings in the List A are contained in the List B then function returns true. For instance, in the Figure A.11 List B does not contain List A. List B contains “Get delivery options”, and “Calculate Delivery time estimates” Strings, but not contain “Login outgoing delivery” string.

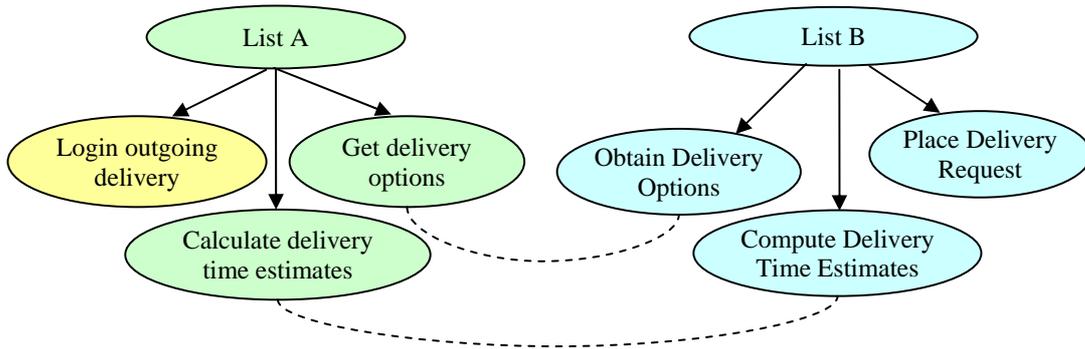


Figure A.11 Using contains function

A.1.2.4 GetAttributeValue function

The `getAttributeValue` function returns the value of an attribute of an XML Element in Saxon. The function receives two parameters a `TinyNodeImpl` node and `String` `attributeName`. The node represents a XML element in the Saxon. For instance, the Figure A.12 shows a `TinyNodeImpl` node in Saxon that represents the object element in XML. If you call the `getAttributeValue` function and pass as parameter node and the `String` "type" then the function returns the value "Agent" as result.

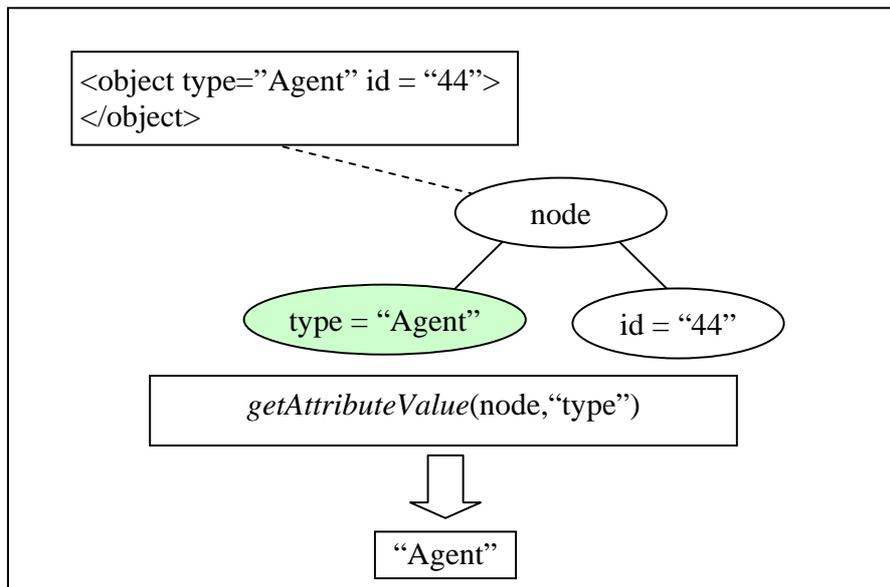


Figure A.12 getAttributeValue function example

A.1.2.5 GetTraceabilityFileName function

The *getTraceabilityFileName* function returns the output filename defined in the project definition and that is used to store the traceability relations.

A.1.2.6 HasRelation function

The *hasRelation* function checks if two elements have a traceability relation. The function receives the id of the elements to be checked and the type of traceability relation. Figure A.13 shows an example when the *hasRelation* function is called to check if elements with id = "*_PsspBiQyEd6fbcmFsKI3Cw*" and id = "40" have a *uses* traceability relation.

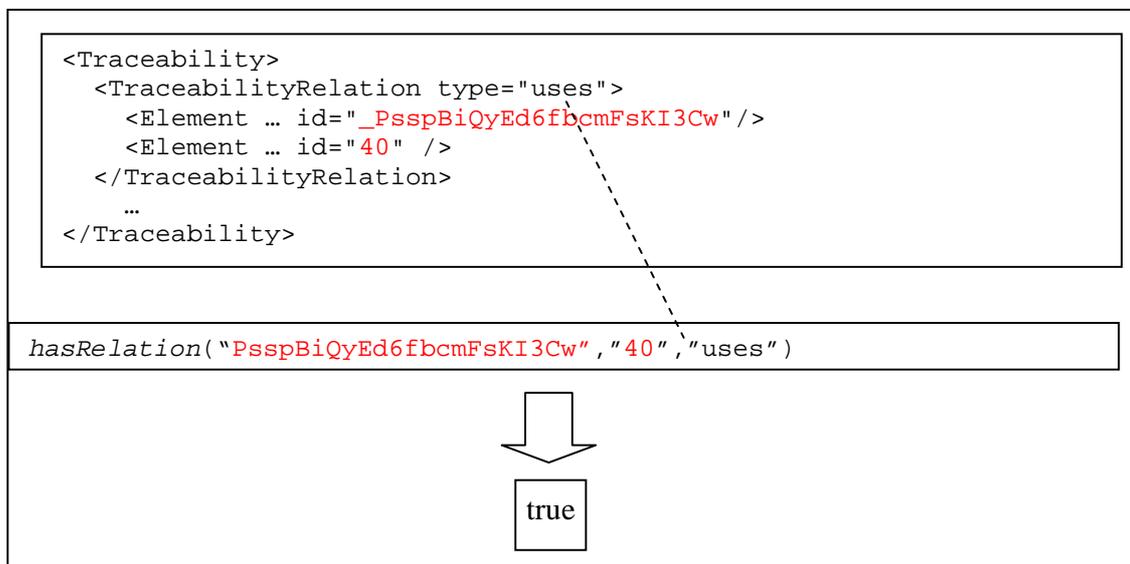


Figure A.13 hasRelation function example

A.1.2.7 stringTokenizer function

The *stringTokenizer* function receives as parameter a String and break down the string into tokens using as delimiters spaces, ‘_’, ‘-’, ‘(’, and ‘)’. In the Figure A.14 a list contained “Get”, “Delivery” and “Options” String is returned as result when function *stringTokenizer* function is called passing as parameter “Get Delivery Options” String.

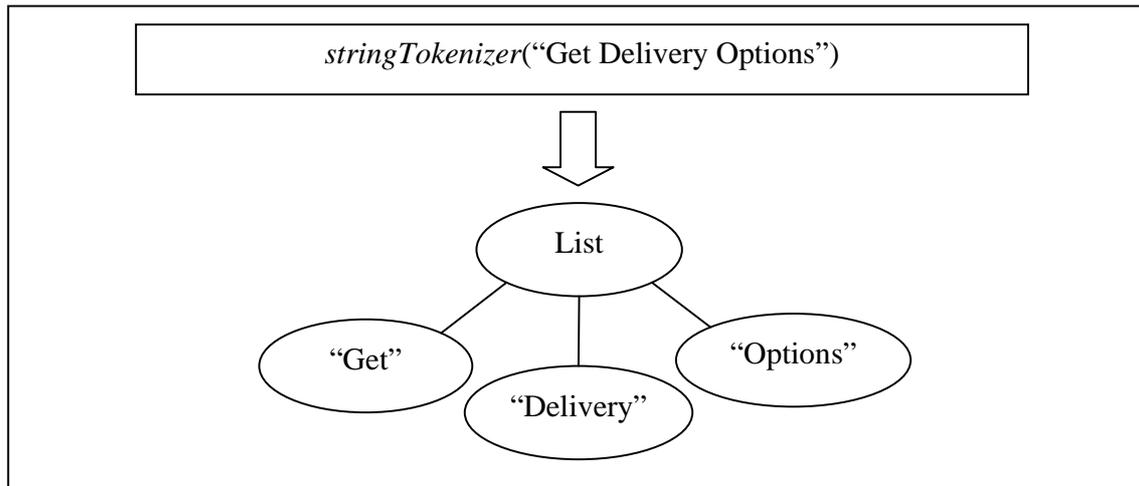


Figure A.14 stringTokenizer function example

A.1.2.8 stringTokenizerByUpperCase function

The *stringTokenizerByUpperCase* function receives as parameter a String and break down the String into tokens using spaces and upper case letters as delimiters. Figure A.15 shows an example when the *stringTokenizerByUpperCase* function is called and “GetDeliveryOptions” String is passed as parameter. As result the stringTokenizerByUpperCase function returns a list of Strings that consists of “Get”, “Delivery” and “Options”.

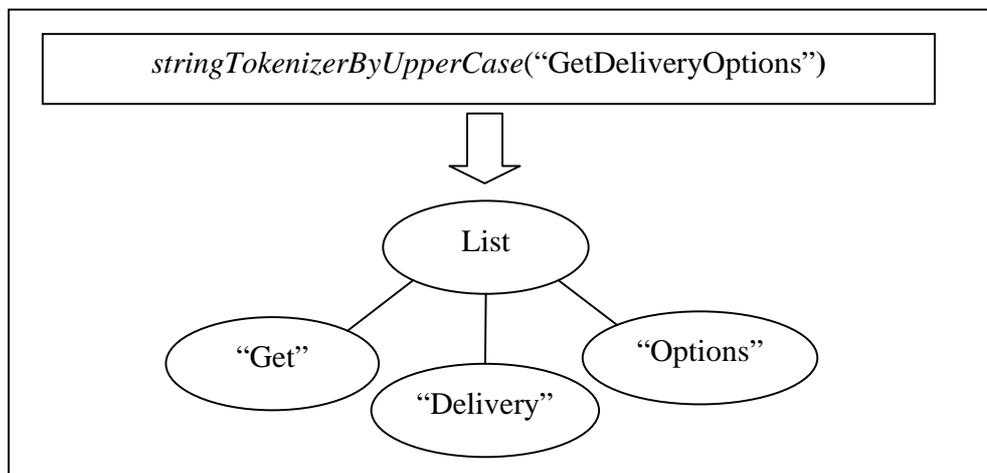


Figure 4.15 stringTokenizerByUpperCase function example

A.1.3 XQueryJACKFunctions

Method Summary	
ArrayList< Field >	getBeliefSetFields (java.lang.String id) It returns an ArrayList with the list of fields of a beliefSet.

String	getJACKFileName() It returns the name of the file that contains the JACK code in XML.
--------	--

Table A.3 XQueryJACKFunctions

A.1.3.1 getBeliefSetFields function

The *getBeliefSetFields* function receives as parameter an id of a beliefSet in Prometheus created using the PDT tool and returns a list of Field elements. Figure A.16 shows an example when the *getBeliefSetFields* function is called and integer 58 is passed as parameter. The *getBeliefSetFields* function returns a list with the included fields.

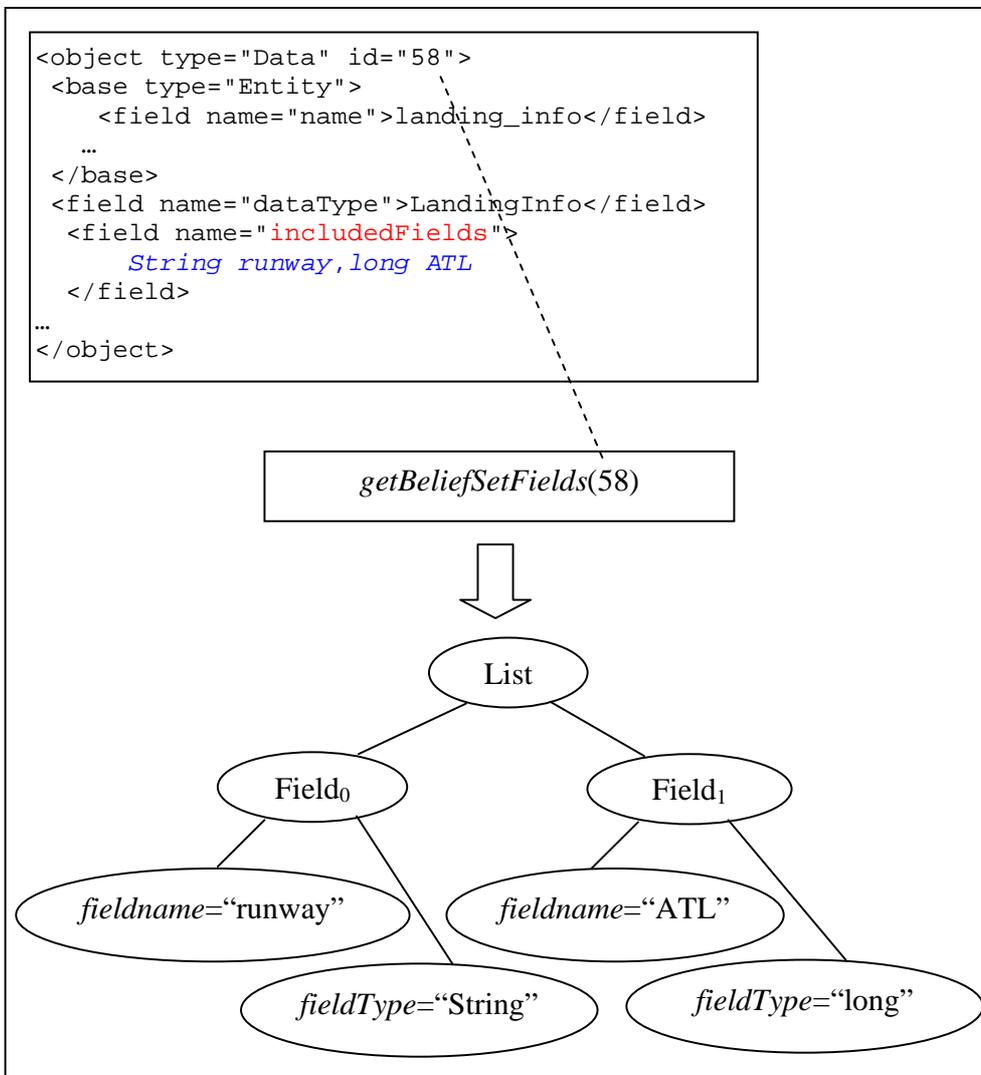


Figure A.16 getBeliefSetFields function example

A.1.3.2 getJACKFileName function

The *getJACKFileName* function returns the filename that contains the xml representation of the JACK code. The JACK filename is defined during the creation of project.

A.1.4 XQueryPDTFunctions

Method Summary	
boolean	actorHasCapability (String actorID, String capabilityID) Finds all agents in Prometheus that uses the Capability and check if there is an overlaps traceability relation between the Actor and the Agent.
Field	fieldTokenizer (String s, String token) Receives a String token and returns a Field object.
String	getAttributeValue (TinyNodeImpl node, String attributeName) Returns the value of an attribute of an XML Element in Saxon
ArrayList< Field >	getIncludedFields (String id) It returns an ArrayList of Fields with the includedFields of a beliefSet in Prometheus.
ArrayList<String>	getInformationCarried (String id) It returns an ArrayList of Strings with informationCarried of a Percept
String	getPDTFileName () Returns the PDT filename
ArrayList< TraceElement >	getPrometheusElements (String id, String type, ArrayList< TraceElement > subElements) It returns an ArrayList of sub-elements of an element in PDT.
ArrayList< TraceElement >	getPrometheusStepScenarios (String id, ArrayList< TraceElement > subElements) Retrieve steps from a Scenario in Prometheus (e.g.
ArrayList< TraceElement >	getPrometheusSubElements (TinyNodeImpl node, String type) It returns an ArrayList of sub-elements of an element in PDT.
ArrayList< TraceElement >	getPrometheusSubGoals (ArrayList< TraceElement > goalsName, String goalID) Retrieve sub-goals of a goal in PDT
ArrayList< TraceElement >	getPrometheusSubGoals (TinyNodeImpl node) Retrieve sub-goals of a goal in the Prometheus model created by the PDT tool
ArrayList< TraceElement >	getPrometheusUsesData (String id, ArrayList< TraceElement > subElements) Retrieves a list of data used by an element (e.g.
boolean	isACapabilityThatTheAgentIncludes (String agentID,

	String capabilityID) Verify if an agent includes a Capability
boolean	<u>isDataProducedByTheRole</u> (String dataID, String roleID) Verifies if a data is produced by a role in Prometheus
boolean	<u>isDataThatTheAgentReads</u> (String dataID, String id) Verifies if an agent reads data in Prometheus
boolean	<u>isDataThatTheAgentWrites</u> (String dataID, String id) Verifies if a agent writes a data in Prometheus
boolean	<u>isDataThatTheCapabilityReads</u> (String dataID, String capabilityID) Verifies if the capability reads a data in Prometheus
boolean	<u>isDataThatTheCapabilityWrites</u> (String dataID, String capabilityID) Verifies if the capability writes data in Prometheus
boolean	<u>isDataThatThePlanReads</u> (String dataID, String planID) Verifies if a data is read by a plan in Prometheus
boolean	<u>isDataThatThePlanWrites</u> (String dataID, String planID) Verifies if a plan writes a data in Prometheus
boolean	<u>isDataUsedByTheRole</u> (String dataID, String roleID) Verifies if a data is used by the role in Prometheus
boolean	<u>isGoalThatTheAgentAchieves</u> (String goalID, String id) Verifies an agent achieves a goal
boolean	<u>isGoalThatTheCapabilityAchieves</u> (String capabilityID, String goalID) Verifies if a capability achieves a goal in Prometheus
boolean	<u>isGoalThatThePlanAchieves</u> (String goalID, String planID) Verifies is a plan achieves a goal
boolean	<u>isAMessageThatTheAgentReceives</u> (String messageID, String agentID) Verifies if an agent receives a message in Prometheus
boolean	<u>isAMessageThatTheAgentSends</u> (String messageID, String agentID) Verifies the agent sends a message in Prometheus
boolean	<u>isAMessageThatTheCapabilityReceives</u> (String messageID, String capabilityID) Verifies the capability receives a message in Prometheus
boolean	<u>isAMessageThatTheCapabilitySends</u> (String messageID, String capabilityID) Verifies if the capability sends a message in Prometheus
boolean	<u>isAMessageThatThePlanReceives</u> (String messageID, String planID) Verifies if a plan receives a message in Prometheus
boolean	<u>isAMessageThatThePlanSends</u> (String messageID,

	String planID) Verifies if the plan sends a message in Prometheus
boolean	<u>isAMessageThatTriggersThePlan</u> (String messageID, String planID) Verifies if the message triggers a plan in Prometheus
boolean	<u>isAnActionThatTheAgentPerforms</u> (String actionID, String id) Verifies an agent performs an action in Prometheus
boolean	<u>isAnActionThatTheCapabilityPerforms</u> (String actionID, String capabilityID) Verifies if a capability performs an action in Prometheus
boolean	<u>isAnActionThatThePlanPerforms</u> (String actionID, String id) Verifies a plan performs an action in Prometheus
boolean	<u>isAPerceptThatTheAgentResponds</u> (String perceptID, String id) Verifies if an agent responds to a percept in Prometheus
boolean	<u>isAPerceptThatTheCapabilityResponds</u> (String perceptID, String id) Verifies if a capability responds to a percept in Prometheus
boolean	<u>isAPerceptThatThePlanResponds</u> (String perceptID, String id) Verifies if the plan responds to the percept in Prometheus
boolean	<u>isAPlanThatTheAgentIncludes</u> (String planID, String id) Verify if an agent includes a plan
boolean	<u>isAPlanThatTheCapabilityIncludes</u> (String planID, String capabilityID) Verifies if a capability includes a plan in Prometheus
boolean	<u>isAPlanTheRoleUses</u> (String planID, String roleID) Verify if role uses a plan in Prometheus
boolean	<u>isARoleThatTheAgentIncludes</u> (String roleID, String id) Verifies is an agent includes a role
boolean	<u>isTrigger</u> (String planId, String eventName) Verifies if a plan triggers an event

Table A.4 XQueryPDTFunctions

A.1.4.1 ActorHasCapability function

The *actorHasCapability* function checks if an actor in i^* has a traceability relation with a capability in Prometheus. The function receives as parameter the id of an actor and the id of a capability then retrieves all agents that implement the capability. If any of the agents that implement the capability has an *overlap* traceability with the actor then the *actorHasCapability* function returns true. In the Figure the *actorHasCapability* function is invoked by passing 44 and , "_8pRv4FyvEd6qIOGYcZQlag" as parameter. The function retrieves all the agents that

includes the capability and then call *isOverlap* function to check if there is an overlaps traceability relation between the Aircraft agent in Prometheus (id = 43) and the Aircraft actor *i**

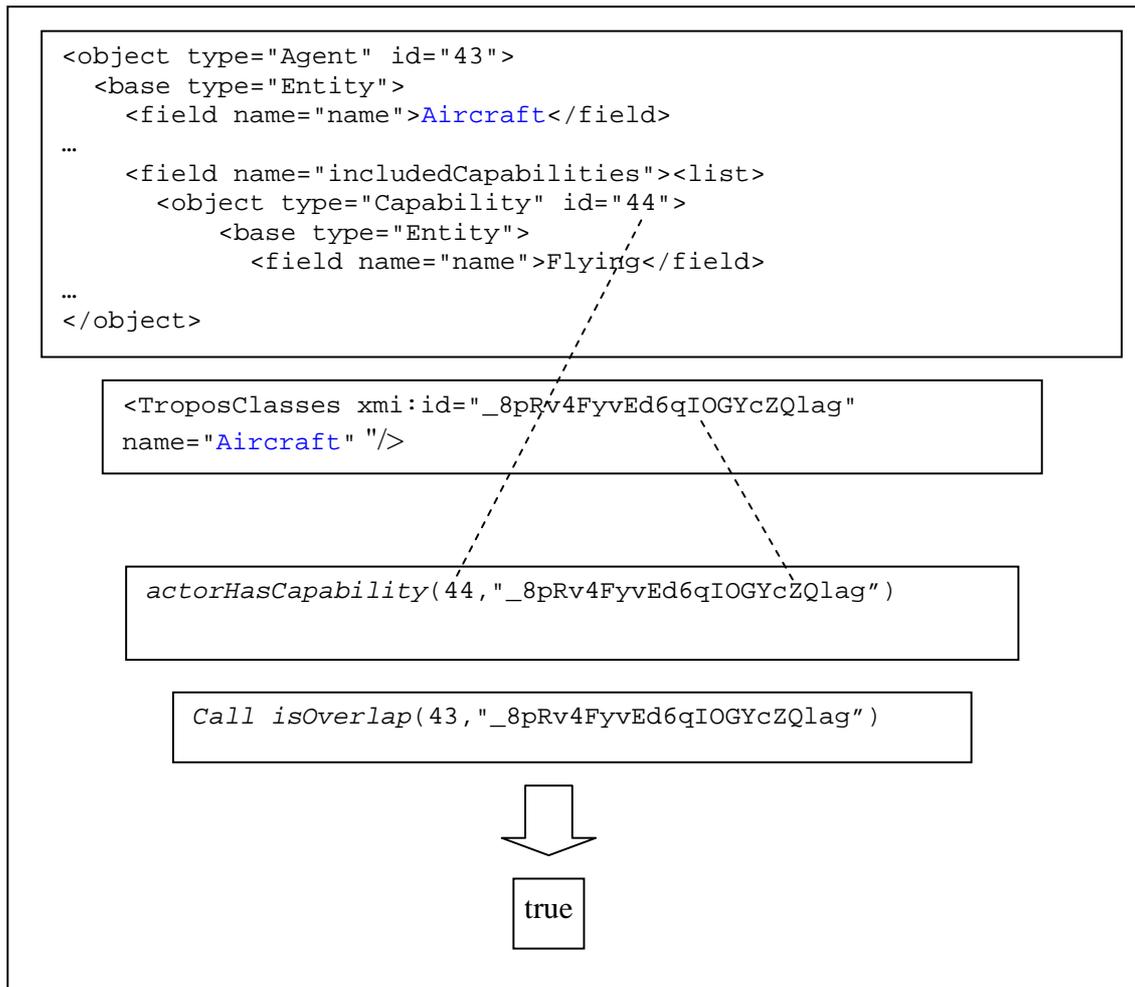


Figure A.17 ActorHasCapability function example

A.1.4.2 FieldTokenizer function

The *fieldTokenizer* function receives two Strings as parameter, *s* that contains information about the field and *token* that contains information how the field is structured and therefore can be divided into tokens. Figure shows an example when the *fieldTokenizer* function is called using as parameter the Strings “long ATL” and “space” as parameter. The *fieldTokenizer* function break down the String using spaces as delimiter and returns a Field element.

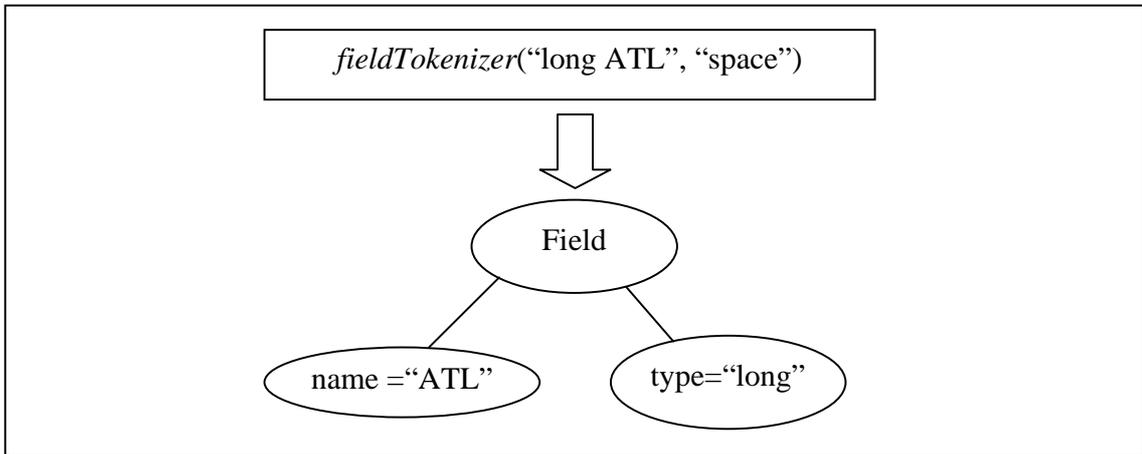


Figure A.18 fieldTokenizer function example

A.1.4.3 GetAttributeValue function

The *getAttributeValue* function returns the value of an attribute of an XML Element in Saxon. The function receives two parameters a *TinyNodeImpl* node and *String* attributeName. The node represents a XML element in the Saxon. For instance, the Figure shows a *TinyNodeImpl* node in Saxon that represents the object element in XML. If you call the *getAttributeValue* function and pass as parameter node and the *String* “type” then the function returns the value “Agent” as result.

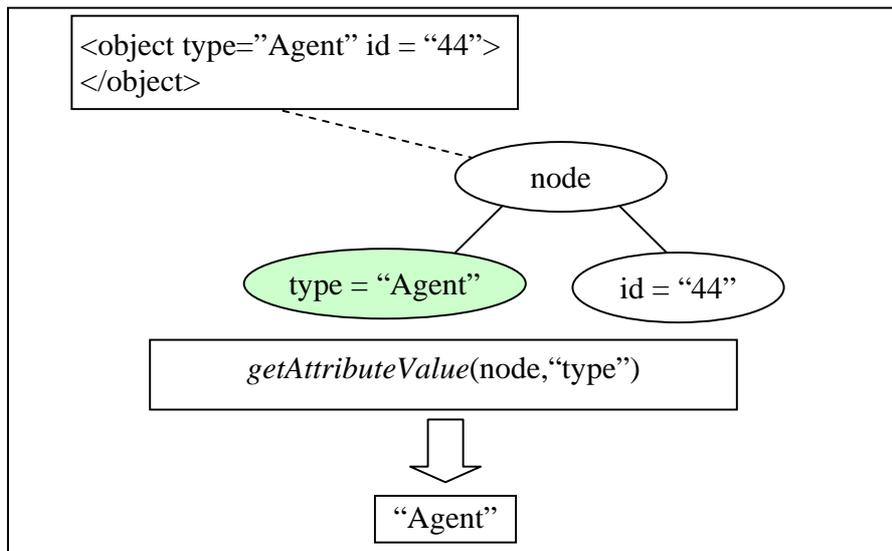


Figure A.19 getAttributeValue function example

A.1.4.4 *GetIncludedFields* function

The *getIncludedFields* function receives the id of an element in Prometheus and returns the included fields Strings as parameter, *s* contains information about the fields and *token* contains information how the fields are structured and can be divided into tokens. The shows an example when the *getIncludedFields* function is called by passing the parameter 35 that is the id of the *runway_info* Data in Prometheus. The function retrieves the information that contains the included fields of the Data and then call the *fieldTokenizer* function. The *fieldTokenizer* function returns a list with the Fields of the Data.

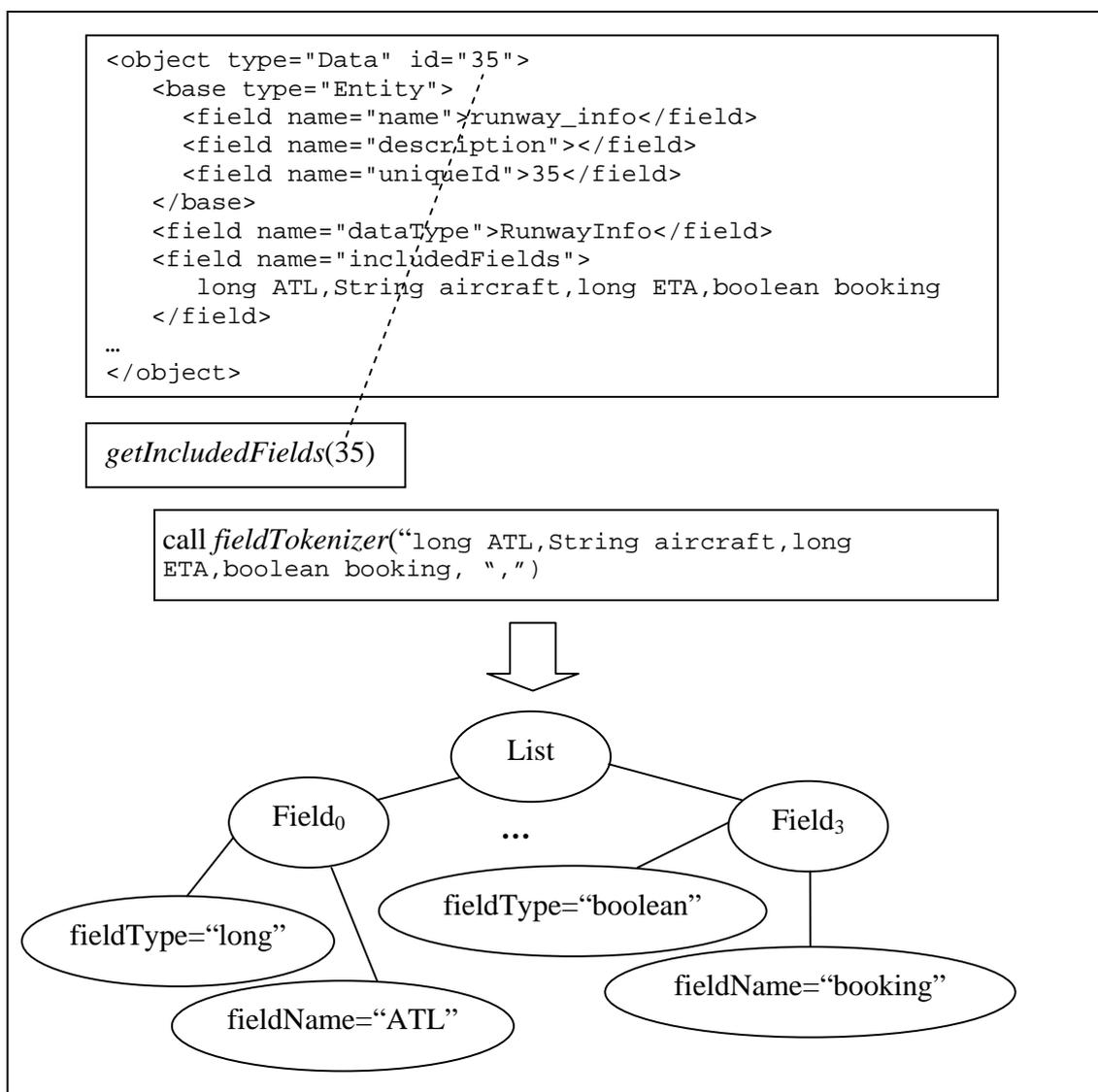


Figure A.20 *getIncludesFields* function example

A.1.4.5 *GetInformationCarried* function

The *getInformationCarried* function receives the id of an element in Prometheus and returns the information carried by the element. Figure A.21 shows an example when the *getInformationCarried* function is called passing as an argument the id equal 7 that is related to the Aircraft Event message. The function retrieves the information carried by the message and returns a list with the Strings “ATL”, and “Slot Allocated”.

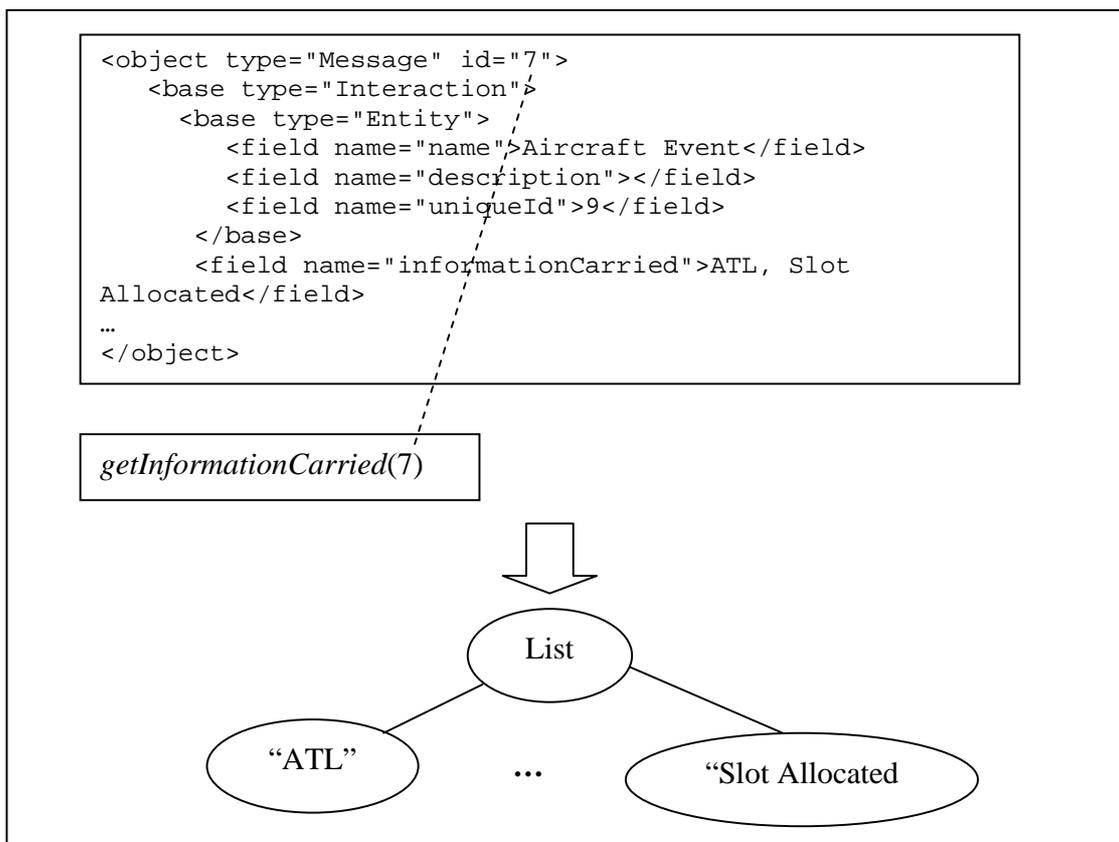


Figure A.21 *getInformationCarried* function example

A.1.4.6 *GetPDTFileName*

The *getPDTFileName* function returns the name of the PDT filename that has been defined during the creation of the project.

A.1.4.7 GetPrometheusElements

The *getPrometheusElements* function receives as parameter a String with the id of an element, a String with the types of sub-elements to be retrieved, and a list with sub-elements in case the function has been called recursively or null. The *getPrometheusElements* function returns an ArrayList of sub-elements of an element in Prometheus created using PDT tool.

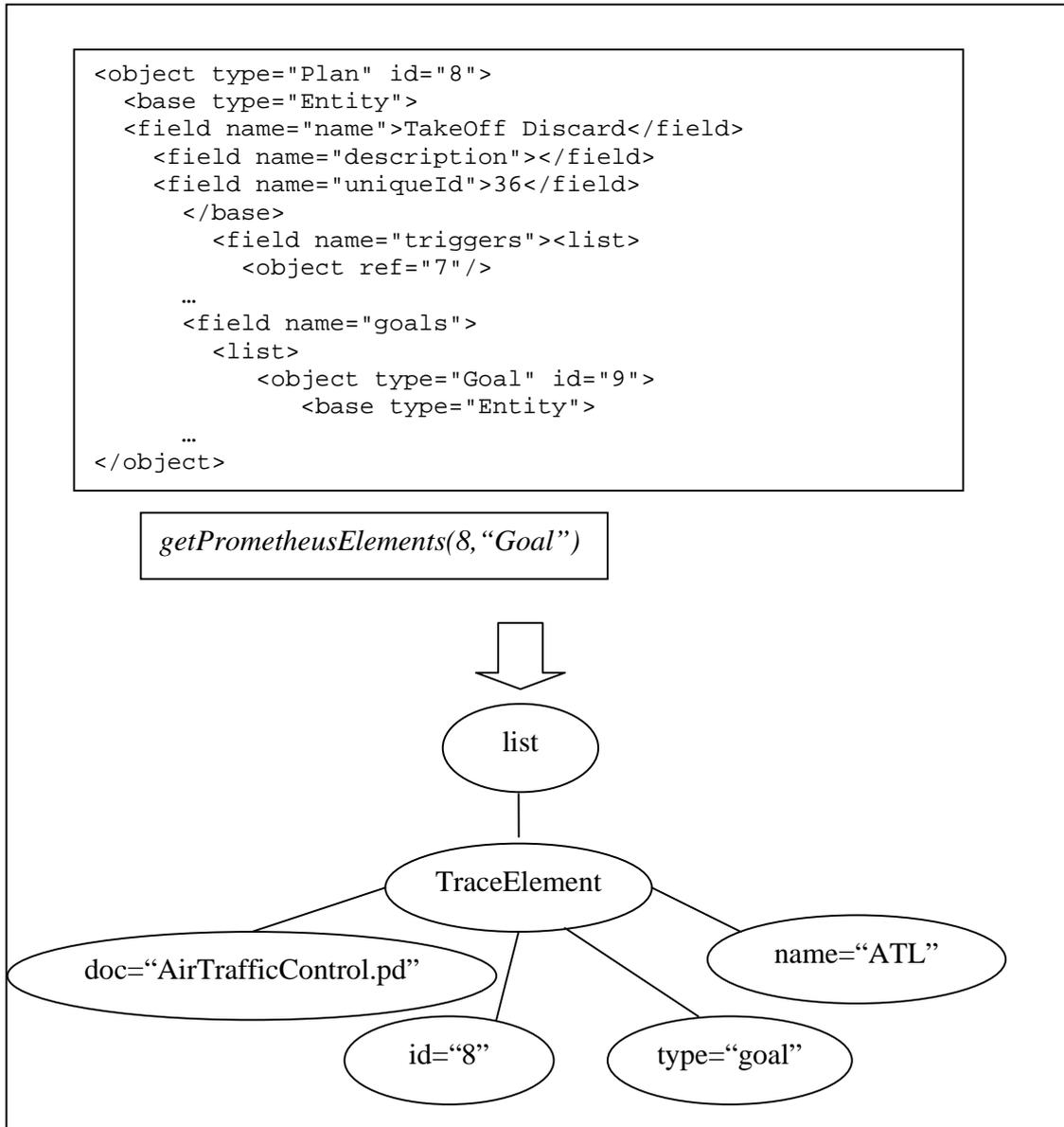


Figure A.22 getPrometheusElements function example

A.1.4.8 GetPrometheusSubElements

The *getPrometheusSubElements* function receives TinyNodeImpl node that represents an XML Element in Prometheus and a String type that represents the type of subelements to be retrieved. If the type is equal to “step” the *getPrometheusStepScenarios* function is called. If the type is equal to “readBy” the *getPrometheusUsesData* function is called, otherwise the *getPrometheusElements* function is called. The function returns a list of sub-elements.

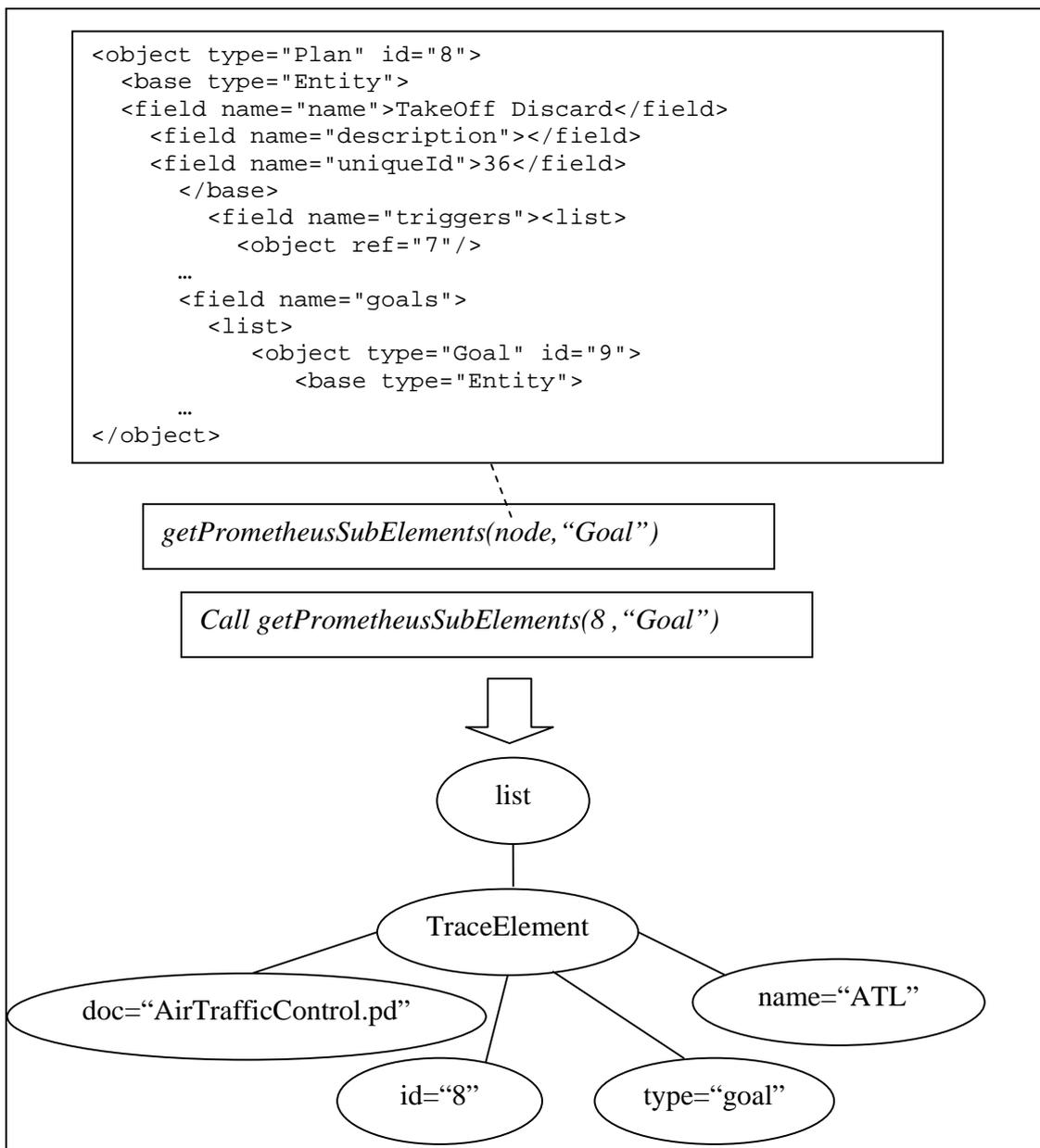


Figure A.23 *getPrometheusSubElements* function example

A.1.4.9 GetPrometheusStepScenarios

The `getPrometheusStepScenarios` function receives as parameter a String with the id of an element and a list of `TraceElement` in case the function has been called recursively, otherwise null.

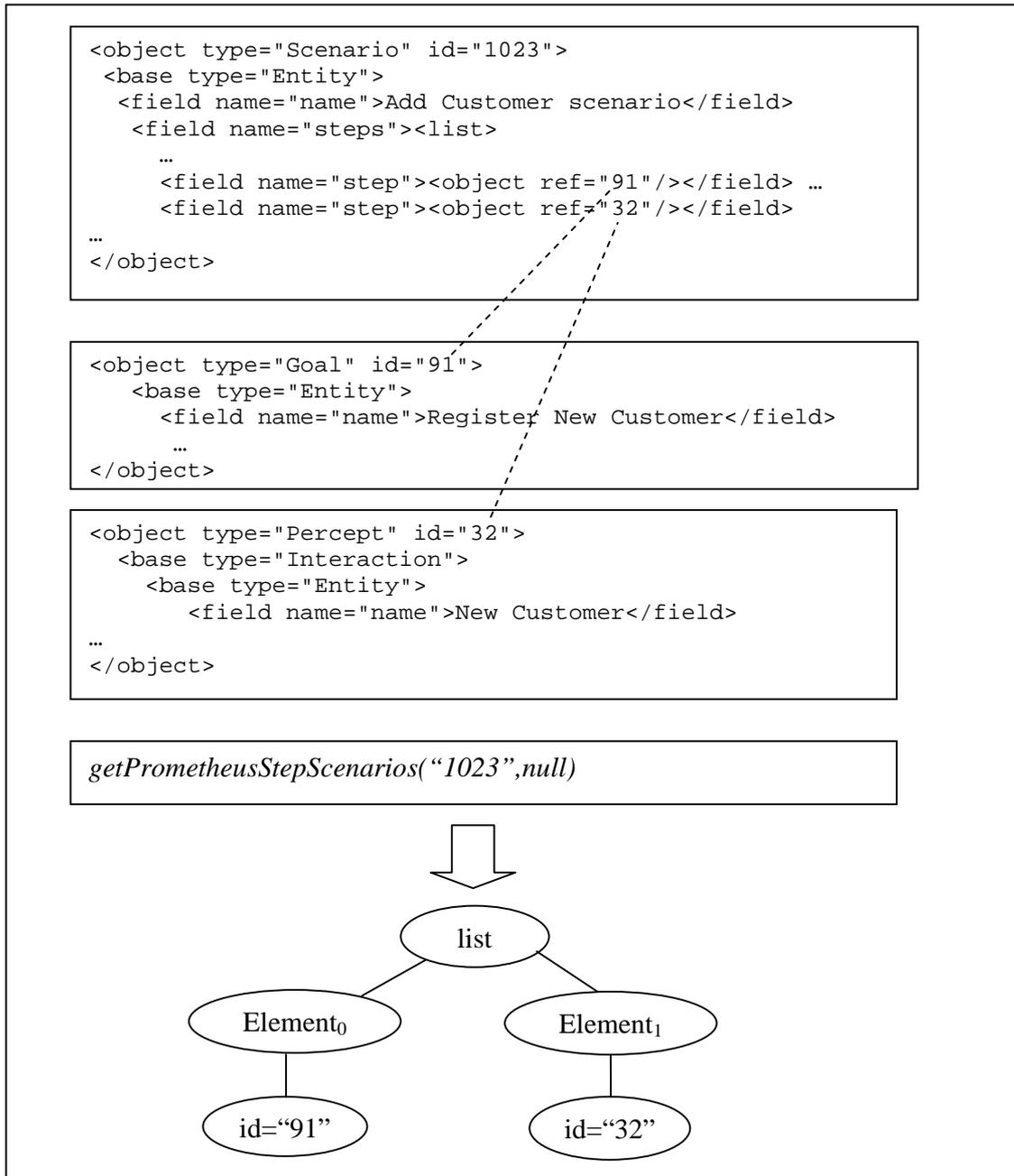


Figure A.24 `getPrometheusStepScenarios` function example

A.1.4.10 GetPrometheusSubGoalsElements

The *getPrometheusSubGoalsElements* function receives as parameter a String with the id of a goal element and a list with sub-elements in case the function has been called recursively otherwise null. The *getPrometheusSubGoalsElements* function returns an ArrayList of sub-goals of an element in Prometheus created using PDT tool.

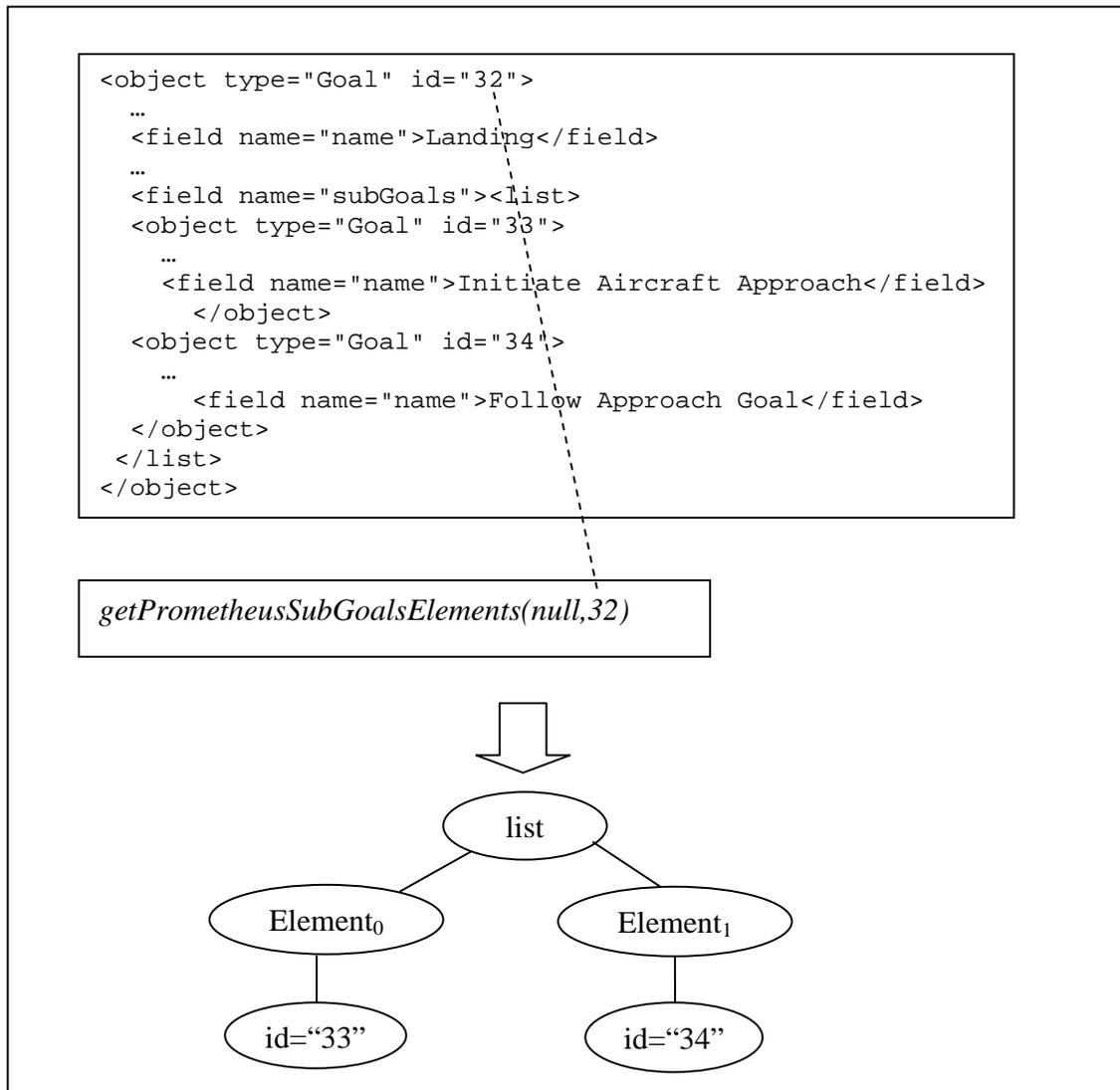


Figure A.25 *getPrometheusSubGoals* function example

A.1.4.11 GetPrometheusSubGoalElements

The *getPrometheusSubGoals* function receives a TinyNodeImpl node that represents a XML Element in Prometheus. The *getPrometheusSubGoals* calls *getPrometheusSubGoalsElements* and pass the id of goal which sub-elements are required. The function returns a list of sub-goals.

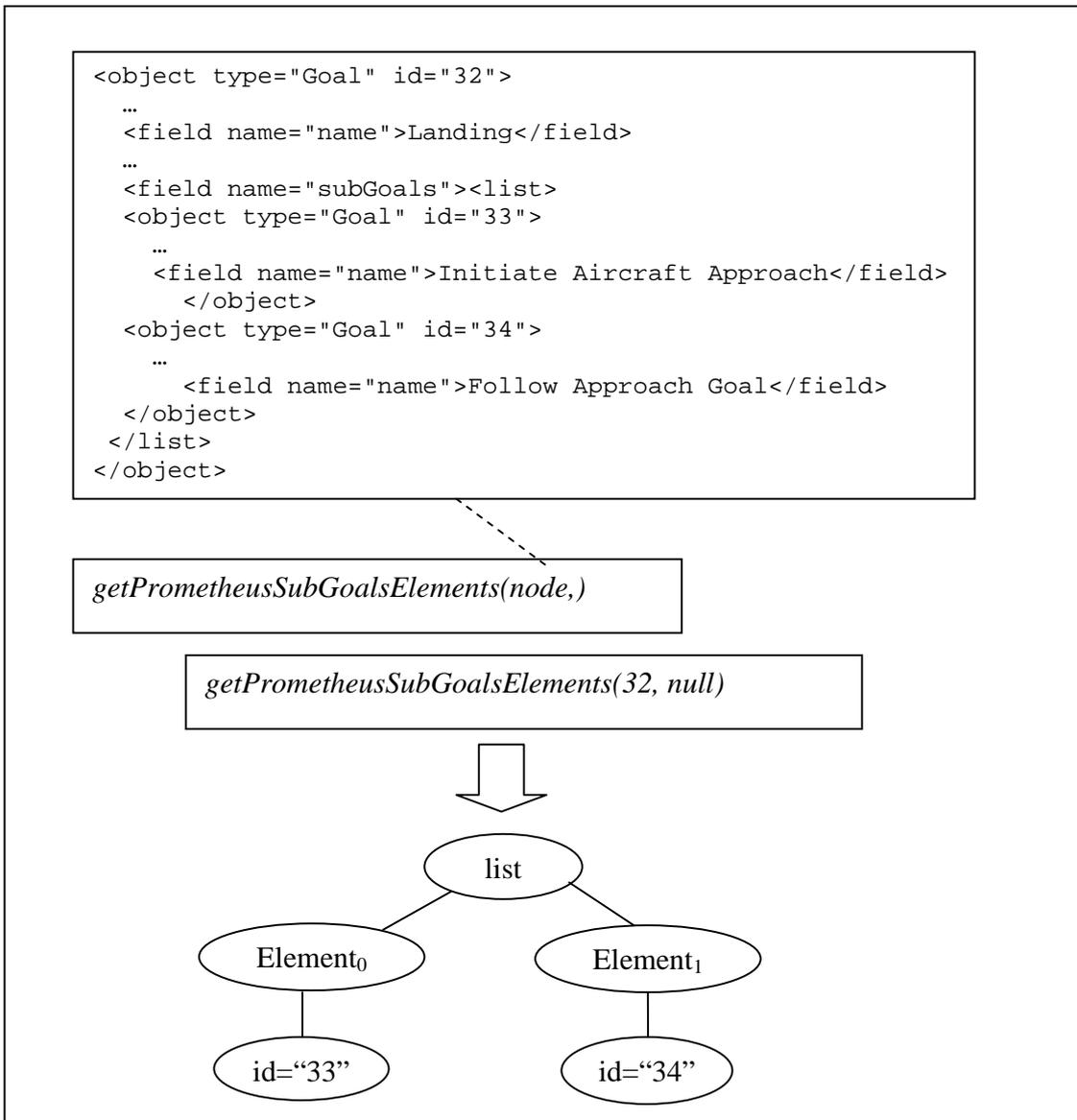


Figure A.26 *getPrometheusSubGoalsElements* function example

A.1.4.12 GetPrometheusUsesData

The *getPrometheusUsesData* function retrieves a list of data used by an element (e.g. Agent, Capability, and Plan) in Prometheus. The *getPrometheusUsesData* function receives a String id as parameter that identifies the element (e.g. Agent, Capability and Plan) and ArrayList of sub-elements that is used when the function is called recursively.

A.1.4.13 IsACapabilityThatTheAgentIncludes

The *isACapabilityThatTheAgentIncludes* function verifies if an agent includes a Capability. The *isACapabilityThatTheAgentIncludes* receives an id that identifies an Agent and the id that identifies the capability. If the agent includes the capability then *isACapabilityThatTheAgentIncludes* function returns true.

A.1.4.14 IsADataProducedByTheRole

The *isADataProducedByTheRole* function verifies if a data is produced by a role in Prometheus. The *isADataProducedByTheRole* function receives as parameter the id of a data in Prometheus, the id of a role in Prometheus. The *isADataProducedByTheRole* function returns true if the data has been produced by the role

A.1.4.15 IsADataThatTheAgentReads

The *isADataThatTheAgentReads* function verifies if a data is read by a plan in Prometheus. The function receives the id of a data in Prometheus and the id of a plan in Prometheus. The *isADataThatTheAgentReads* function returns true if the data is read by a plan.

A.1.4.16 IsADataThatTheAgentWrites

The *isADataThatTheAgentWrites* function verifies if an agent writes a data in Prometheus. The *isADataThatTheAgentWrites* function receives as parameter the id of data in Prometheus and the id of an agent in Prometheus. The *isADataThatTheAgentWrites* function returns true if the agent writes a data.

A.1.4.17 IsADataThatTheCapabilityReads

The *isADataThatTheCapabilityReads* function verifies if the capability reads a data in Prometheus. The *isADataThatTheCapabilityReads* function receives a parameter the id of data in Prometheus and the id of capability in Prometheus. The *isADataThatTheCapabilityReads* function returns true if the capability reads a data.

A.1.4.18 IsADataThatTheCapabilityWrites

The *isADataThatTheCapabilityWrites* function verifies if the capability writes a data in Prometheus. The *isADataThatTheCapabilityWrites* function receives as parameter the id of a data and the id of a capability. The *isADataThatTheCapabilityWrites* function returns true if the capability writes the data.

A.1.4.19 IsADataThatThePlanReads

The *isADataThatThePlanReads* function verifies if a data is read by a plan in Prometheus. The *isADataThatThePlanReads* function receives as parameter the id of a data in Prometheus and the id of a plan in Prometheus. The *isADataThatThePlanReads* function returns true if the data is read by the plan.

A.1.4.20 IsADataThatThePlanWrites

The *isADataThatThePlanWrites* function verifies if a plan writes a data in Prometheus. The function receives as parameter the id of data in Prometheus and the id of a plan in Prometheus. The *isADataThatThePlanWrites* function returns true if the plan writes the data.

A.1.4.21 IsADataUsedByTheRole

The *isADataUsedByTheRole* function verifies if a data is used by the role in Prometheus. The *isADataUsedByTheRole* function receives the id of a data in Prometheus and the id of a role in Prometheus. The *isADataUsedByTheRole* function returns true if the data is used by the role.

A.1.4.22 IsADataThatTheAgentAchieves

The *isADataTheAgentAchieves* function verifies if an agent writes a data in Prometheus. The *isADataTheAgentAchieves* function receives as parameter the id of a data in Prometheus and the

id of an agent in Prometheus. The *isADataTheAgentAchieves* function returns true if the agent writes the data

A.1.4.23 IsAGoalThatTheCapabilityAchieves

The *isAGoalThatTheCapabilityAchieves* function verifies if a capability achieves a goal in Prometheus. The *isAGoalThatTheCapabilityAchieves* function receives the id of the capability in Prometheus and the id of a goal in Prometheus. The *isAGoalThatTheCapabilityAchieves* function returns true if the capability achieves the goal.

A.1.4.24 IsAGoalThatThePlanAchieves

The *isAGoalThatThePlanAchieves* function verifies if a plan achieves a goal. The *isAGoalThatThePlanAchieves* function receives the id of a goal in Prometheus and the id of a plan in Prometheus. The *isAGoalThatThePlanAchieves* function returns true if the goal is achieved by the plan.

A.1.4.25 IsAGoalThatTheAgentAchieves

The *isGoalThatTheAgentAchieves* function verifies if an agent includes a role. The *isGoalThatTheAgentAchieves* function receives as parameter the id of a role in Prometheus and the id of an agent in Prometheus. The function returns true if the agent includes the role.

A.1.4.26 IsAGoalThatTheCapabilityAchieves

The *isGoalThatTheCapabilityAchieves* function verifies if the capability achieves a goal in Prometheus. The *isGoalThatTheCapabilityAchieves* function receives as parameter the id of the capability in Prometheus and the id of a goal in Prometheus. The *isGoalThatTheCapabilityAchieves* returns true if the capability achieves the goal.

A.1.4.27 IsAMessageThatTheAgentReceives

The *isAMessageThatTheAgentReceives* function verifies if an agent receives a message in Prometheus. The *isAMessageThatTheAgentReceives* function receives the id of a message in Prometheus and the id of an agent in Prometheus. The *isAMessageThatTheAgentReceives* function returns true if the agent receives the message.

A.1.4.28 IsAMessageThatTheAgentSends

The *isAMessageThatTheAgentSends* function verifies if an agent sends a message in Prometheus. The *isAMessageThatTheAgentSends* function receives the id of the message in Prometheus and the id of an agent in Prometheus. The *isAMessageThatTheAgentSends* function returns true if the agent sends the message

A.1.4.29 IsAMessageThatTheCapabilityReceives

The *isAMessageThatTheCapabilityReceives* function verifies if a capability in Prometheus receives a message in Prometheus. The *isAMessageThatTheCapabilityReceives* function receives the id of a message in Prometheus and the id of a capability in Prometheus. The *isAMessageThatTheCapabilityReceives* function returns true if the capability receives the message.

A.1.4.30 IsAMessageThatTheCapabilitySends

The *isAMessageThatTheCapabilitySends* function verifies if the capability sends a message in Prometheus. The *isAMessageThatTheCapabilitySends* function receives as parameter the id of a message in Prometheus and the id of a capability in Prometheus. The *isAMessageThatTheCapabilitySends* function returns true if the capability sends a message.

A.1.4.31 IsAMessageThatTheReceives

The *isAMessageThatTheCapabilitySends* function verifies if a plan receives a message in Prometheus. The *isAMessageThatTheCapabilitySends* function receives the id of a message in Prometheus and the id of a plan in Prometheus. The *isAMessageThatTheCapabilitySends* function returns true if the plan receives the message.

A.1.4.32 IsAMessageThatThePlanReceives

The *isAMessageThatThePlanReceives* function verifies if a plan receives a message in Prometheus. The *isAMessageThatThePlanReceives* function receives an id of a message in Prometheus and the id of a plan in Prometheus. The *isAMessageThatThePlanReceives* function returns true if the plan receives the message.

A.1.4.33 *IsAMessageThatThePlanSends*

The *isAMessageThatThePlanSends* function verifies if the plan sends a message in Prometheus. The *isAMessageThatThePlanSends* function receives the id of a message in Prometheus and the id of a plan in Prometheus. The *isAMessageThatThePlanSends* function returns true if the plan sends the message.

A.1.4.34 *IsAMessageThatTriggersThePlan*

The *isAMessageThatTriggersThePlan* function verifies if the message triggers a plan in Prometheus. The *isAMessageThatTriggersThePlan* function receives the id of a message in Prometheus and the id of a plan in Prometheus. The *isAMessageThatTriggersThePlan* function returns true if the message triggers the plan.

A.1.4.35 *IsAnActionThatTheAgentPerforms*

The *isAnActionThatTheAgentPerforms* function verifies if an agent performs an action in Prometheus. The *isAnActionThatTheAgentPerforms* function receives the id of an action in Prometheus and the id of an agent in Prometheus. The *isAnActionThatTheAgentPerforms* returns true if the agent performs the action.

A.1.4.36 *IsAnActionThatTheCapabilityPerforms*

The *isAnActionThatTheCapabilityPerforms* function verifies if a capability performs an action in Prometheus. The *isAnActionThatTheCapabilityPerforms* function receives an id of an action in Prometheus and the id of a capability in Prometheus. The *isAnActionThatTheCapabilityPerforms* function returns true if the capability performs the action.

A.1.4.37 *IsAnActionThatThePlanPerforms*

The *isAnActionThatThePlanPerforms* function verifies if a plan performs an action in Prometheus. The *isAnActionThatThePlanPerforms* function receives the id of an action in

Prometheus and the id of plan in Prometheus. The *isAnActionThatThePlanPerforms* function returns true if the agent performs the action.

A.1.4.38 IsAPerceptThatTheAgentResponds

The *isAPerceptThatTheAgentResponds* function verifies if an agent responds to a percept in Prometheus. The *isAPerceptThatTheAgentResponds* function receives the id of a percept in Prometheus and the id of an agent in Prometheus. The *isAPerceptThatTheAgentResponds* function returns true if the agent responds to the percept.

A.1.4.39 IsAPerceptThatTheCapabilityResponds

The *isAPerceptThatTheCapabilityResponds* function verifies if a capability responds to a percept in Prometheus. The *isAPerceptThatTheCapabilityResponds* function receives the id of a percept in Prometheus and the id of a capability in Prometheus. The *isAPerceptThatTheCapabilityResponds* function returns true if the capability responds to the percept.

A.1.4.40 IsAPerceptThatThePlanResponds

The *isAPerceptThatThePlanResponds* function verifies if the plan responds to the percept in Prometheus. The *isAPerceptThatThePlanResponds* function receives the id of a percept in Prometheus and the id of a plan in Prometheus. The *isAPerceptThatThePlanResponds* function returns true if the plan responds to the percept.

A.1.4.41 IsAPerceptThatTheCapabilityResponds

The *isAPerceptThatTheCapabilityResponds* function verifies if a capability responds to a percept in Prometheus. The *isAPerceptThatTheCapabilityResponds* function receives the id of a percept in Prometheus and the id of a capability in Prometheus. The *isAPerceptThatTheCapabilityResponds* function returns true if the capability responds to the percept.

A.1.4.42 IsAPerceptThatThePlanResponds

The *isAPerceptThatThePlanResponds* function verifies if the plan responds to the percept in Prometheus. The *isAPerceptThatThePlanResponds* function receives the id of a percept in Prometheus and the id of a plan in Prometheus. The *isAPerceptThatThePlanResponds* function returns true if the plan responds to the percept.

A.1.4.43 IsAPerceptThatTheCapabilityResponds

The *isAPerceptThatTheCapabilityResponds* function verifies if a capability responds to a percept in Prometheus. The *isAPerceptThatTheCapabilityResponds* function receives the id of a percept in Prometheus and the id of a capability in Prometheus. The *isAPerceptThatTheCapabilityResponds* function returns true if the capability responds to the percept.

A.1.4.44 IsAPerceptThatThePlanResponds

The *isAPerceptThatThePlanResponds* function verifies if the plan responds to the percept in Prometheus. The *isAPerceptThatThePlanResponds* function receives the id of a percept in Prometheus and the id of a plan in Prometheus. The *isAPerceptThatThePlanResponds* function returns true if the plan responds to the percept.

A.1.4.45 IsAPlanThatTheAgentIncludes

The *isAPlanThatTheAgentIncludes* function verifies if an agent includes a plan in Prometheus. The *isAPlanThatTheAgentIncludes* function receives the id of a plan in Prometheus and id of an agent in Prometheus. The *isAPlanThatTheAgentIncludes* function returns true if the agent includes the plan.

A.1.4.46 IsAPlanThatTheCapabilityIncludes

The *isAPlanThatTheCapabilityIncludes* function verifies if a capability includes a plan in Prometheus. The *isAPlanThatTheCapabilityIncludes* function receives the id of a plan in Prometheus and the id of a capability in Prometheus. The *isAPlanThatTheCapabilityIncludes* function returns true if the capability includes the plan.

A.1.4.47 IsAPlanTheRoleUses

The *isAPlanTheRoleUses* function verifies if a role in Prometheus uses a plan in Prometheus. The *isAPlanTheRoleUses* function receives the id of a plan in Prometheus and the id of a role I Prometheus. The *isAPlanTheRoleUses* function returns true if the role uses the plan.

A.1.4.48 IsARoleThatTheAgentIncludes

The *isARoleThatTheAgentIncludes* function verifies is an agent in Prometheus includes a role in Prometheus. The *isARoleThatTheAgentIncludes* function receives as parameter the id of a role and the id of an agent. The *isARoleThatTheAgentIncludes* function returns true if the agent includes the role.

A.1.4.49 IsTrigger

The *isTrigger* function verifies if a plan triggers an event. The *isTrigger* function receives the id of a plan in Prometheus and the name of an event. The *isTrigger* function returns true if the event triggers the plan.

A.1.5 XQuerySimilarityFunctions

Method Summary	
boolean	<u>creates</u> (String idA, String idB) Verifies if there is a creates traceability relation between two elements
boolean	<u>creates</u> (String idA, String idB, Document outputDoc, XPath xpath) Verifies if there is a creates traceability relation between two elements
boolean	<u>hasUses</u> (String idA, String idB) Verifies if there is uses traceability relation between two elements
boolean	<u>hasUses</u> (String idA, String idB, Document outputDoc, XPath xpath) Verifies if there is an uses traceability relation between two elements
boolean	<u>isOverlap</u> (String idA, String idB)

	Verify if two elements has an overlaps traceability relation
boolean	isOverlap (String idA, String idB, Document outputDoc, XPath xpath) Check if two elements has an overlaps relation
boolean	isPositiveSimilar (ArrayList< TraceElement > list1, ArrayList< TraceElement > list2, double threshold)
boolean	isSimilar (java.util.ArrayList< TraceElement > list1, java.util.ArrayList< TraceElement > list2, double threshold)
boolean	isSimilarAgentAgent (TinyNodeImpl node1, TinyNodeImpl node2, double threshold)
int	isSimilarByOverlaps (ArrayList< TraceElement > list1, ArrayList< TraceElement > list2)
boolean	isSimilarByOverlaps (ArrayList< TraceElement > list1, ArrayList< TraceElement > list2, double threshold)
boolean	isSimilarCapabilityCapability (TinyNodeImpl node1, TinyNodeImpl node2, double threshold)
boolean	isSimilarDataAndBeliefSet (TinyNodeImpl node1, TinyNodeImpl node2, double threshold)
boolean	isSimilarPlanPlan (TinyNodeImpl node1, TinyNodeImpl node2, double threshold)
boolean	isSimilarSDResourceAndMessage (TinyNodeImpl node1, String node2)
boolean	someOverlap (java.lang.String elem, java.util.ArrayList< TraceElement > list)

Table A.5 XQuerySimilarityFunctions

A.1.5.1 HasUses function

The *hasUses* function check if two elements have an *uses* traceability relation. The function receives the id of the elements to be compared, the document that contains the traceability relation and an XPath object to execute the evaluation. Figure 4.30 shows an example when the *hasUses* function is called to check if elements with id = "*_PsspBiQyEd6fbcmFsKI3Cw*" and id = "40" have a *uses* traceability relation.

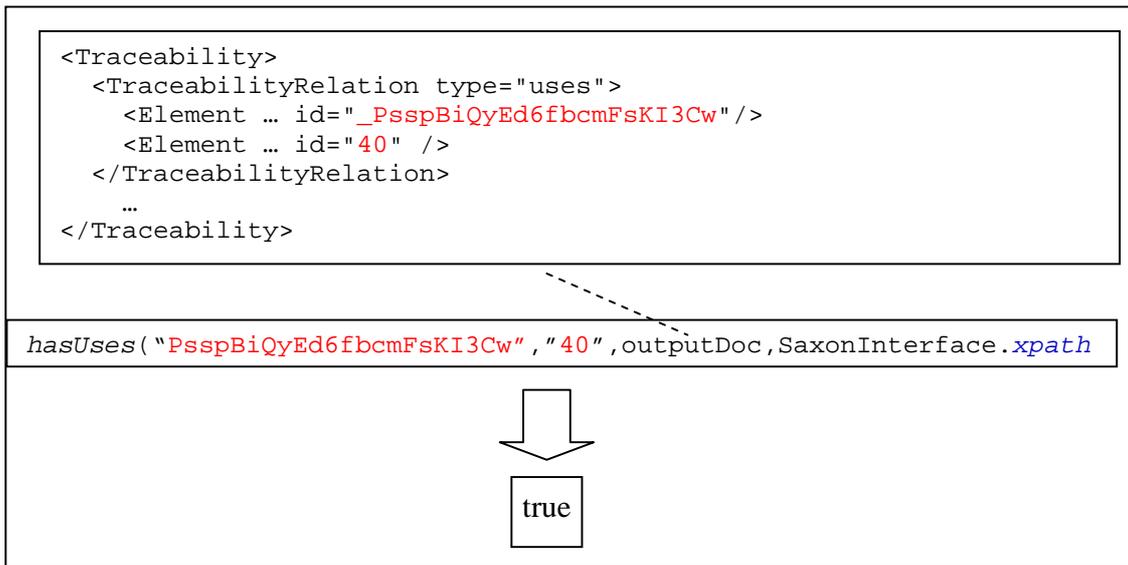


Figure A.27 hasUses function example

A.1.5.2 HasUses function

The *hasUses* function check if two elements have an *uses* traceability relation. The function receives the id of the elements to be compared. The *hasUses function* call *hasUses* function explained in the Section A.1.5.1. Figure A.28 shows an example when the *hasUses* function is called to check if elements with id = "_PsspBiQyEd6fbcMfsKI3Cw" and id = "40" have a *uses* traceability relation.

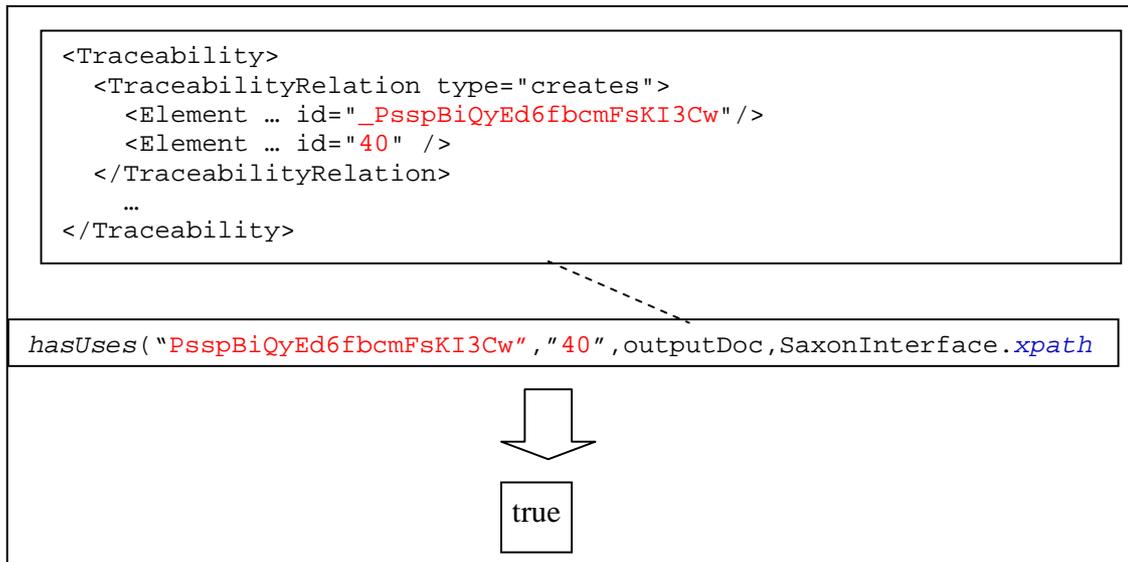


Figure A.28 hasUses function example

A.1.5.3 Creates function

The *creates* functions check if two elements have a creates traceability relation. The function receives the id of the elements to be compared, the document that contains the traceability relations and XPath object to execute the evaluation. Figure A.29 shows an example when the *creates* function is called to check if elements with id = "_PsspBiQyEd6fbcMfsKI3Cw" and id = "40" have a creates traceability relation.

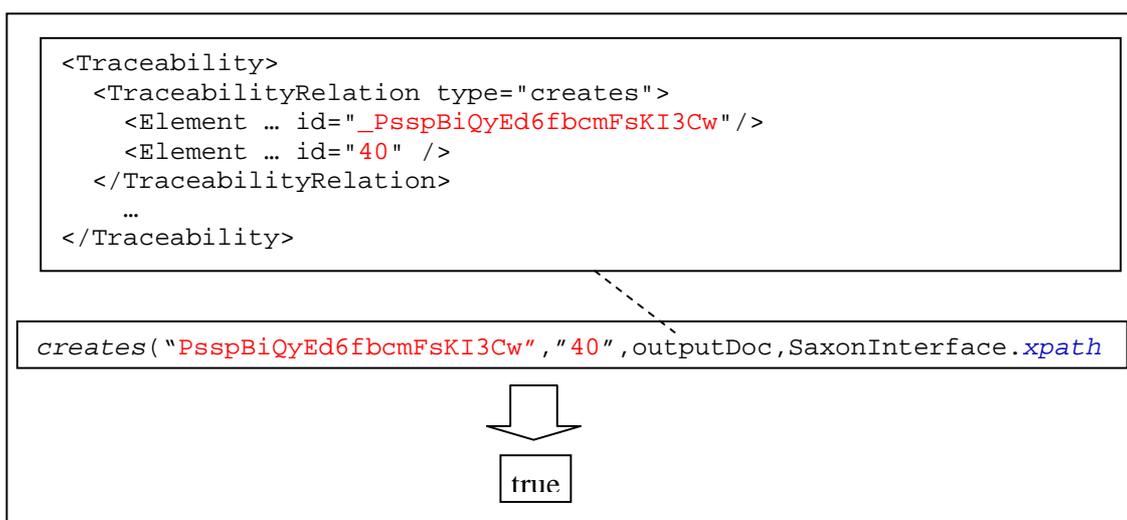


Figure A.29 creates function example

A.1.5.4 Creates function

The *creates* function check if two elements have a *creates* traceability relation. The function receives the id of the elements to be compared. The *creates function* call *creates* function explained in the Section A.1.5.3. Figure A.30 shows an example when *creates* function is called to check if elements with id = "*PsspBiQyEd6fbcMfsKI3Cw*" and id = "40" have a *creates* traceability relation.

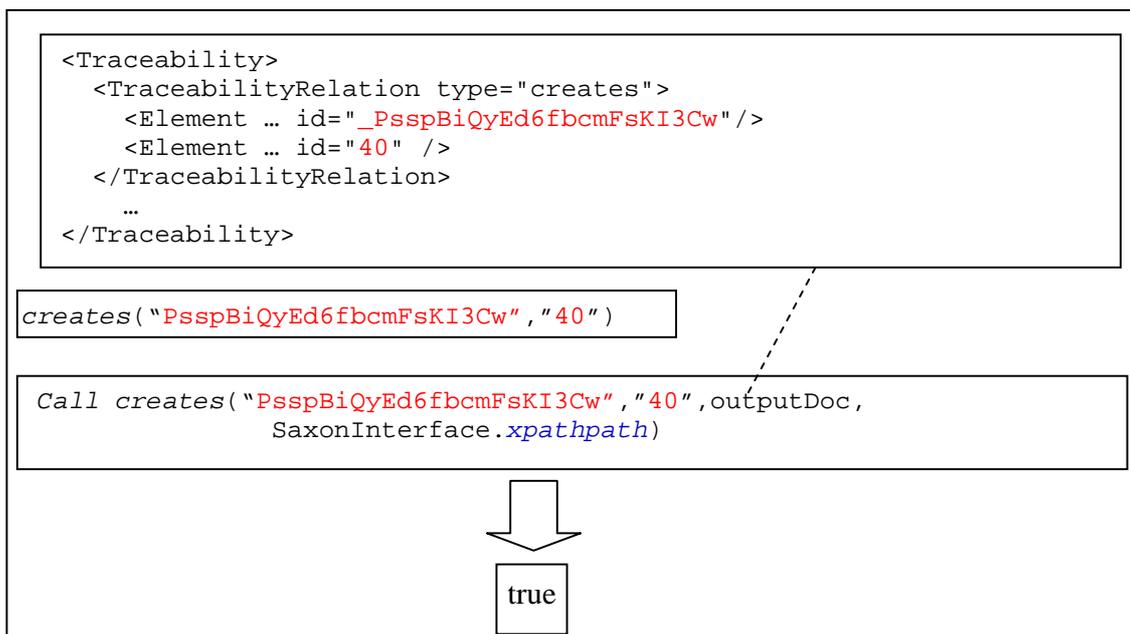


Figure A.30 creates function example

A.1.5.5 IsOverlap function

The *overlaps* functions check if two elements have a *creates* traceability relation. The function receives the id of the elements to be compared, the document that contains the traceability relation and a XPath object to execute the evaluation. Figure A.31 shows an example when the *overlaps* function is called to check if elements with id = "*PsspBiQyEd6fbcMfsKI3Cw*" and id = "40" have *isOverlap* traceability relation.

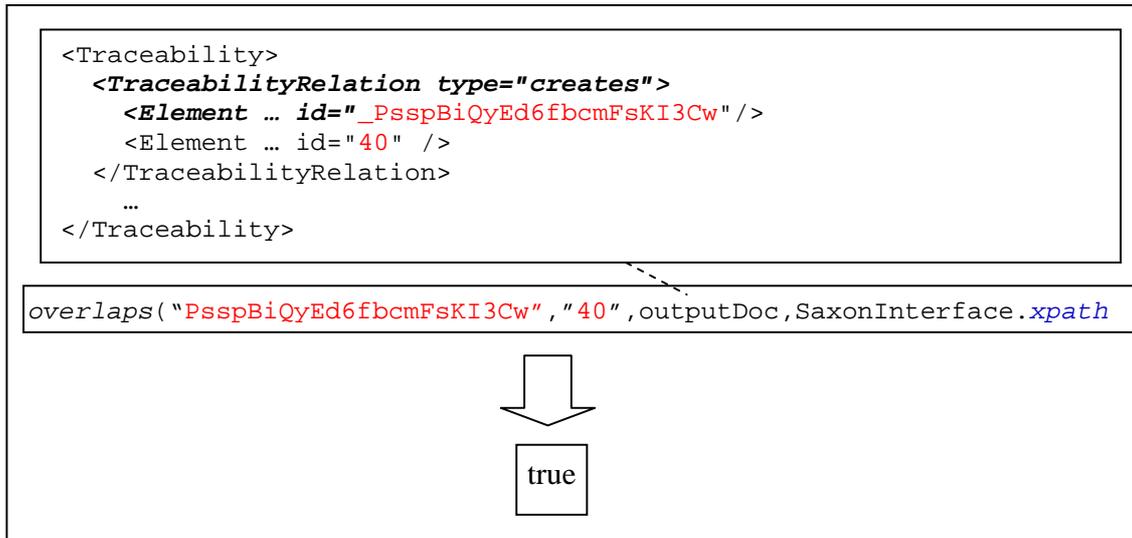


Figure A.31 overlaps function example

A.1.5.6 *IsOverlap* function

The *isOverlap* function checks if two elements have an *overlaps* traceability relation. The function receives the id of the elements to be compared. The *isOverlap* function call is *isOverlap* function explained in the Section A.1.5.5. Figure shows an example when the function is called to check if elements with id = `"_PsspBiQyEd6fbcMfsKI3Cw"` and id = `"40"` have *isOverlap* traceability relation.

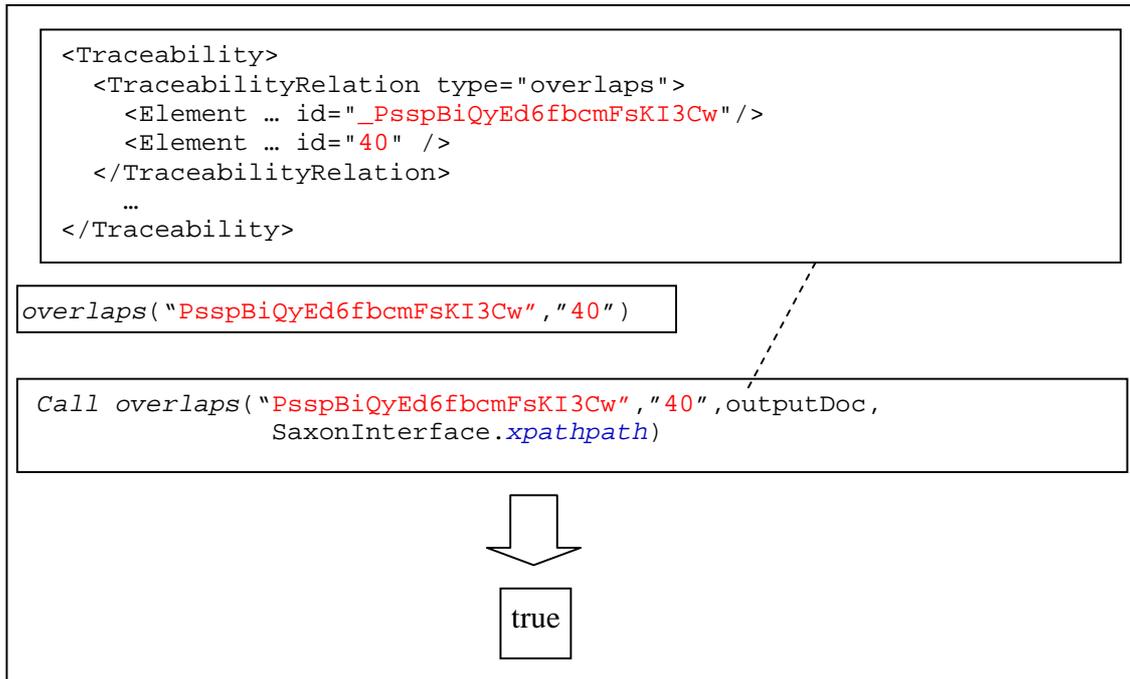


Figure A.32 overlaps function example

A.1.5.7 IsPositiveSimilar function

The *isPositiveSimilar* function receives two lists of elements and compare if the number of elements in the list1 which names are synonyms to elements in the list2 is greater than a threshold. If list1 is empty then *isPositiveSimilar* function returns true. In the Figure A.33, the *isPositiveSimilar* function is called passing list1 and list2 as parameter. The function compares if the name of each element in the list1 has a synonyms in the list2. The only elements in the list1 that does not have a synonym in the list2 is “Login outgoing delivery” therefore the percentage of elements in the list1 that has a synonym in the list2 is 66.7 that is greater than 40 (threshold).

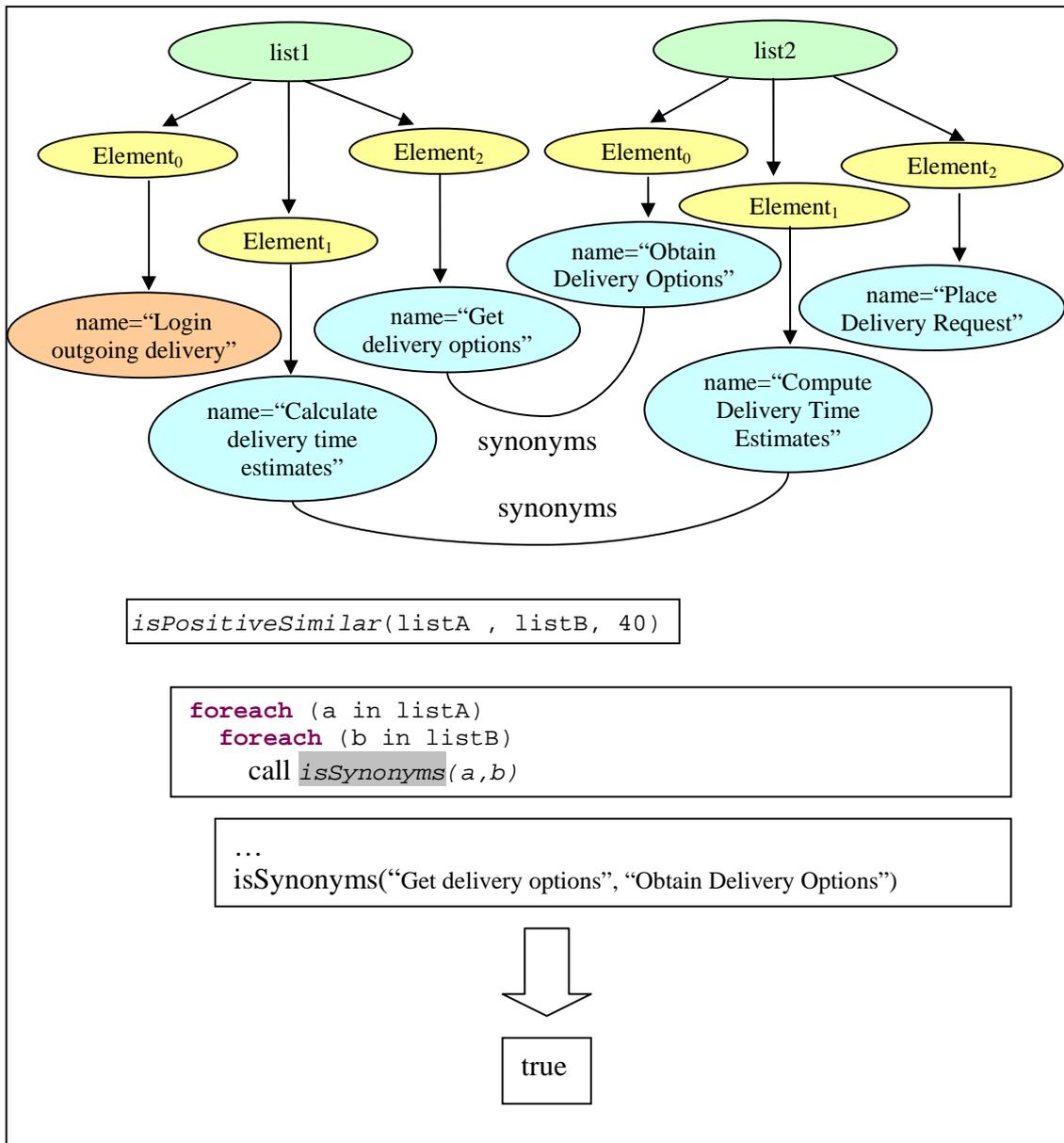


Figure A.33 isPositiveSimilar function example

A.1.5.8 IsSimilar function

The isSimilar function receives two lists of elements and compare if the number of elements in the *list1* which names are synonyms to elements in the *list2* is greater than a threshold. If the *list1* is empty then *isSimilar* function returns true. In the Figure A.34, the *isSimilar* function is called passing list1 and list2 as parameter. The function compares if the name of each element in the list1 has a synonyms in the list2. The only element in the list1 that does not have a

synonym in the list2 is “Login outgoing delivery” therefore the percentage of elements in the list1 that has a synonym in the list2 is 66.7 that is greater than 40 (threshold).

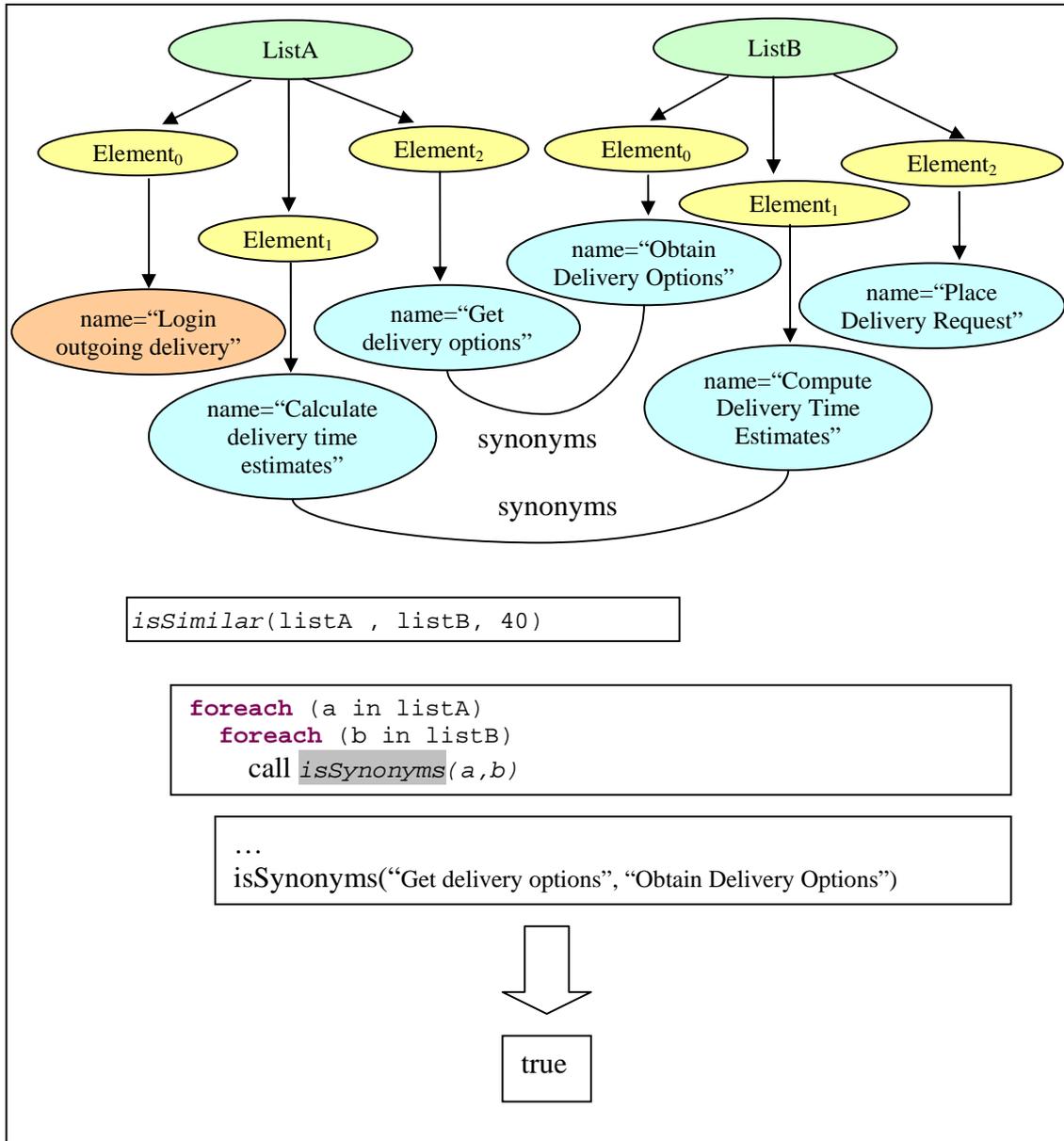


Figure A.34 isSimilar function example

A.1.5.9 IsSimilarByOverlaps

The isSimilarByOverlaps function receives two lists of elements and returns the number of elements in the list1 that has an overlaps traceability relation with any element in the list2.

The `isSimilarByOverlaps` function call `isOverlap` function explained in the A.1.5.6 section. If an element in the list1 does not have an *overlaps* relation then the element is added to the list of missing elements.

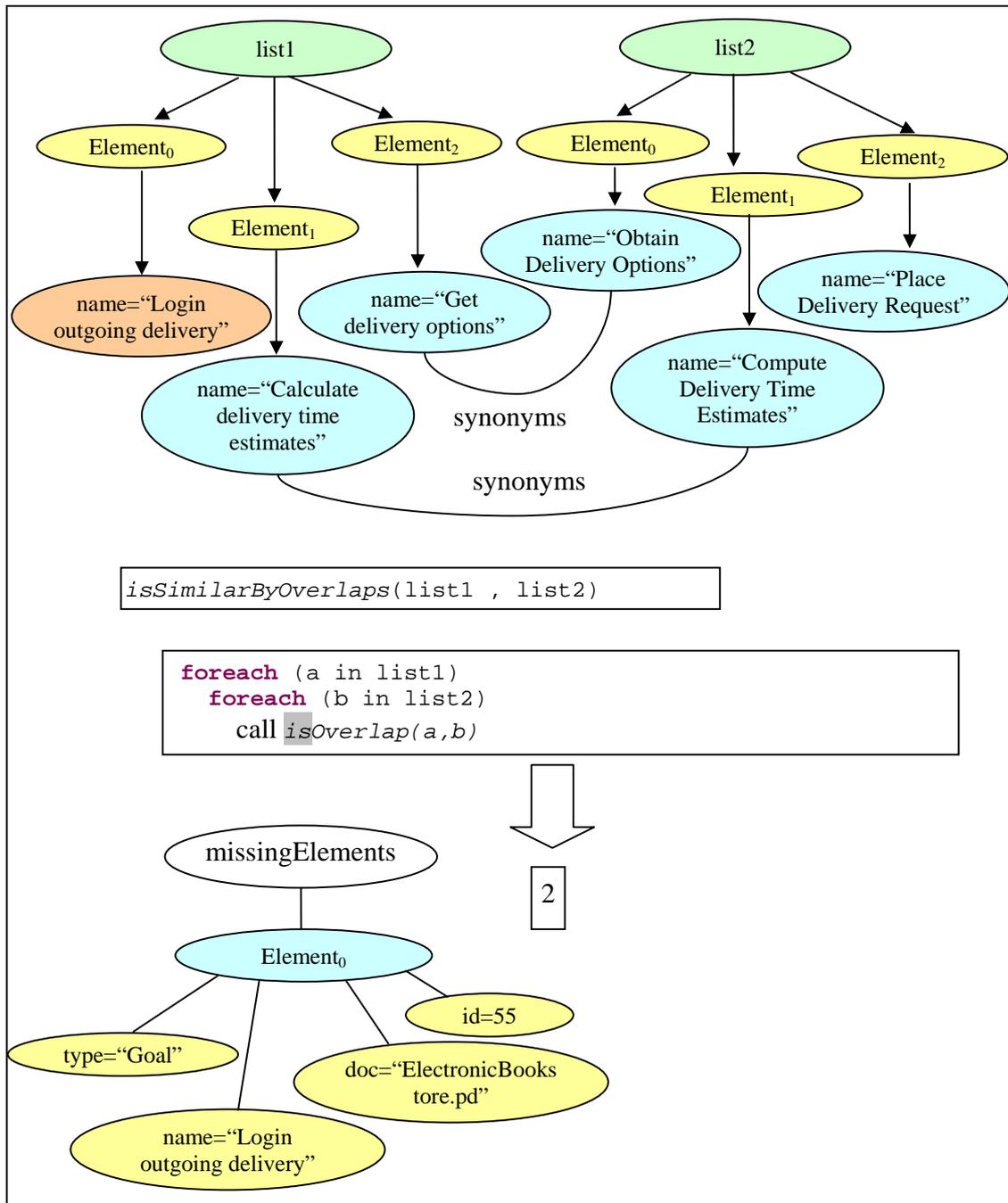


Figure A.35 `isSimilarByOverlaps` function example

A.1.5.10 IsSimilarDataAndBeliefSet

The *isSimilarDataAndBeliefSet* function receives two TinyNodeImpl nodes that represent a Prometheus data in XML and a JACK beliefSet in XML. The function compares if a data and a beliefset are similar based on the name of the fields of the data and the beliefset. If the percentage of the name of the fields that are synonyms are greater than a threshold then the method returns true.

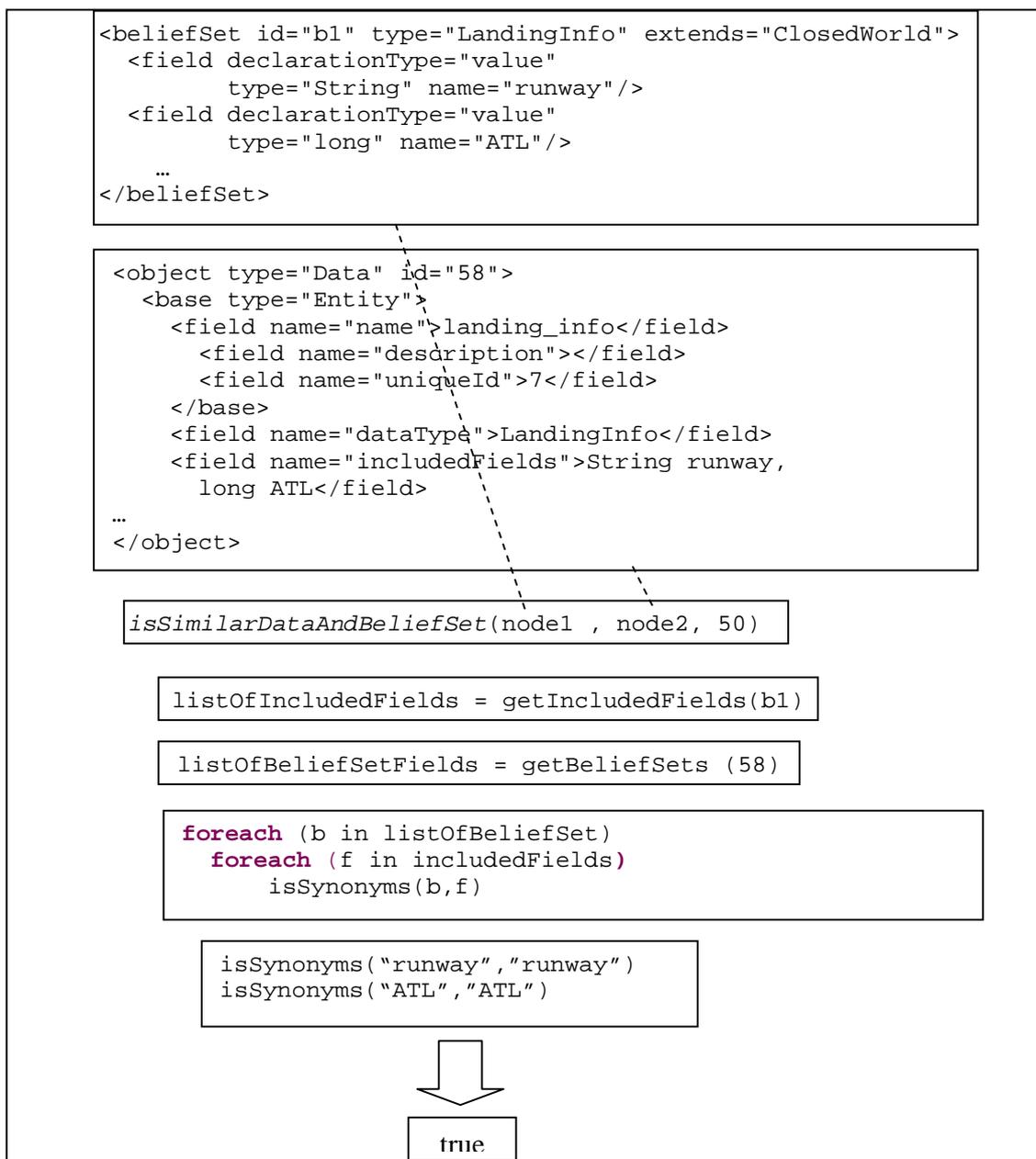


Figure A.36 *isSimilarDataAndBeliefSet* function example

A.1.5.11 *isSimilarSDResourceAndMessage*

The *isSimilarSDResourceAndMessage* function receives a `TinyNodeImpl` node that represents a SD Resource and a `String` with the id of the Prometheus message. The *isSimilarSDResourceAndMessage* function calls *getInformationCarried* function to retrieve the information carried by the message. Then the *isSimilarSDResourceAndMessage* function calls *isSynonyms* function (explained in the A.1.6.3 Section) to check if the information carried by the message is synonyms to the name of the `SDResource`.

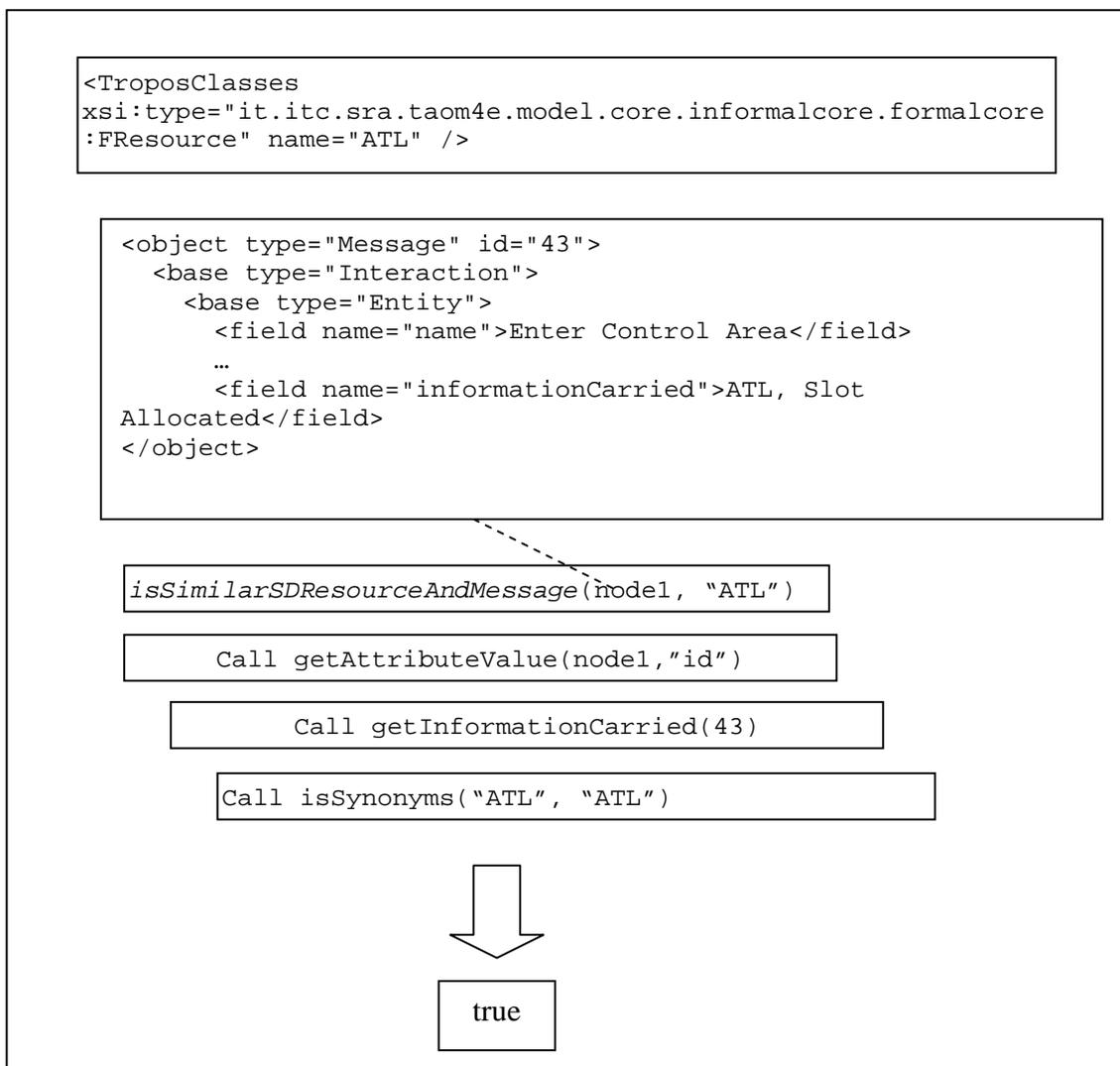


Figure A.37 *isSimilarSDResourceAndMessage* function example

A.1.5.12 SomeOverlap

The *someOverlap* function receives a String *id* that represents the identifier of an element and a list of *TraceElement*. The function verifies if there is any overlaps traceability relation between the element identified by *id* and any of the elements in the *TraceElement* list.

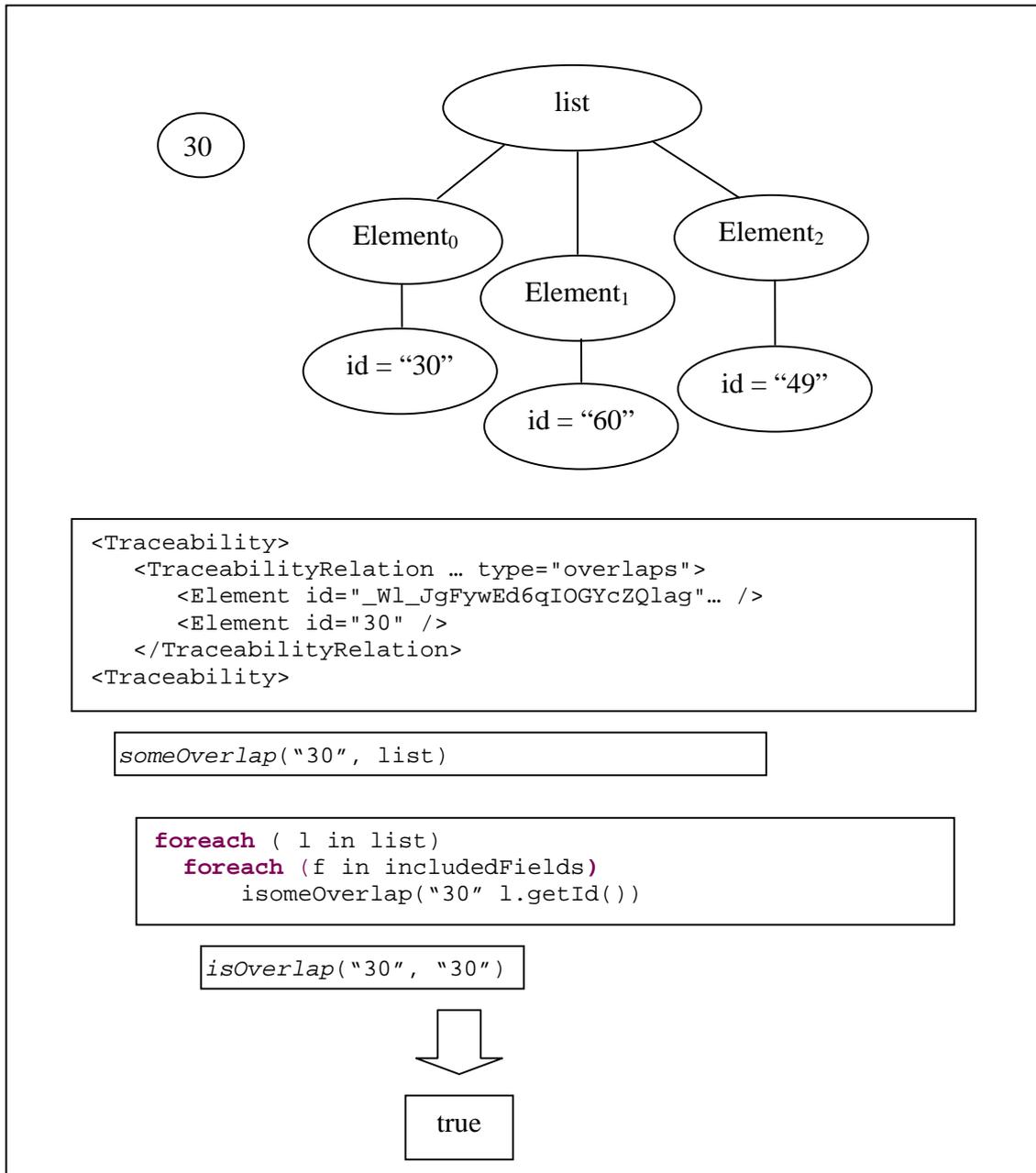


Figure A.38 *SomeOverlap* function example

A.1.6 XQuerySynonymsFunctions

Method Summary	
boolean	contains (ArrayList<String> list1, ArrayList<String> list2)
boolean	contains (String word, ArrayList<String> wordList)
boolean	isSynonyms (String str1, String str2)
ArrayList<String>	stringTokenizer (String str)
ArrayList<String>	stringTokenizerByUpperCase (String str)

Table A.6 XQuerySynonyms Function example

A.1.6.1 StringTokenizerByUpperCase function

The *stringTokenizerByUpperCase* function receives as parameter a String *str* and it breaks down the string into tokens using spaces and upper case letters as delimiters. Figure A.39 shows an example when the “GetDeliveryOptions” String is passed as parameter to the *stringTokenizerByUpperCase* function. As result, the *stringTokenizerByUpperCase* function returns a list of strings that consists of “Get”, “Delivery” and “Options”.

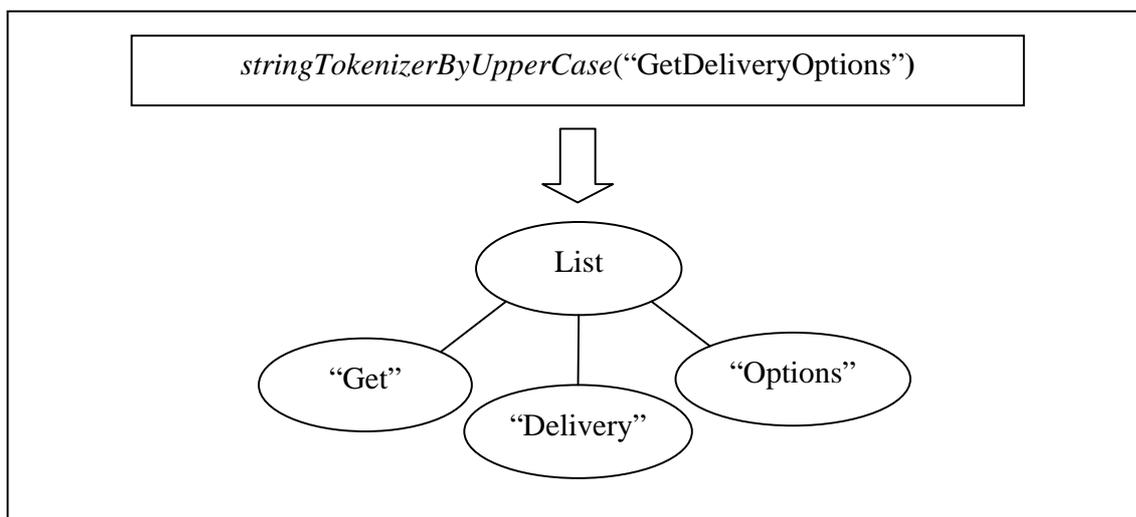


Figure A.39 stringTokenizerByUpperCase function example

A.1.6.2 Contains function

The *contains* function receives as parameter a String *word* and an ArrayList of String *wordList* and check if the ArrayList *wordList* contains the String *word*. The function uses the WordNet dictionary to check for synonyms words. Figure A.40 shows that the *contains* function returns true when it is invoked passing as parameter the String “Get” and the list of Strings *wordList* that consists of “Obtain”, “Delivery” and “Options”. The list of Strings *wordList* contains the “Get” string because “Get” and “Obtain” are synonyms.

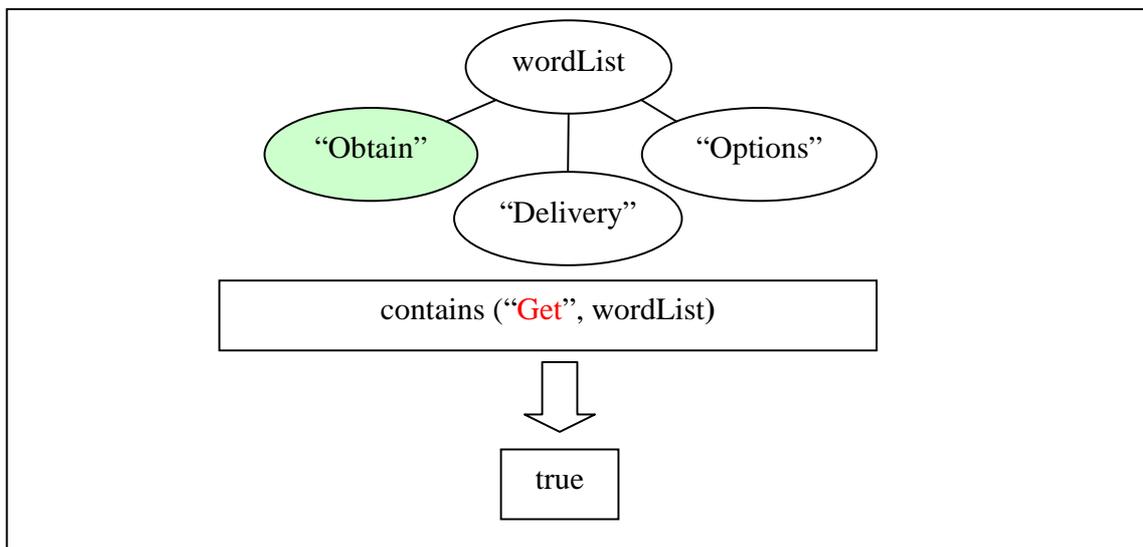


Figure A.40 contains function example

A.1.6.3 IsSynonyms function

The *isSynonyms* function receives as parameter two Strings *str1* and *str2*. The *isSynonyms* function uses the *stringTokenizerByUpperCase* (see A.1.6.1 section) function to break *str1* and *str2* in two lists of words, *wordList1* and *wordList2*. The *contains* function (see A.1.6.2 Section) is used to verify if *wordList2* contains each word in *wordList1*.

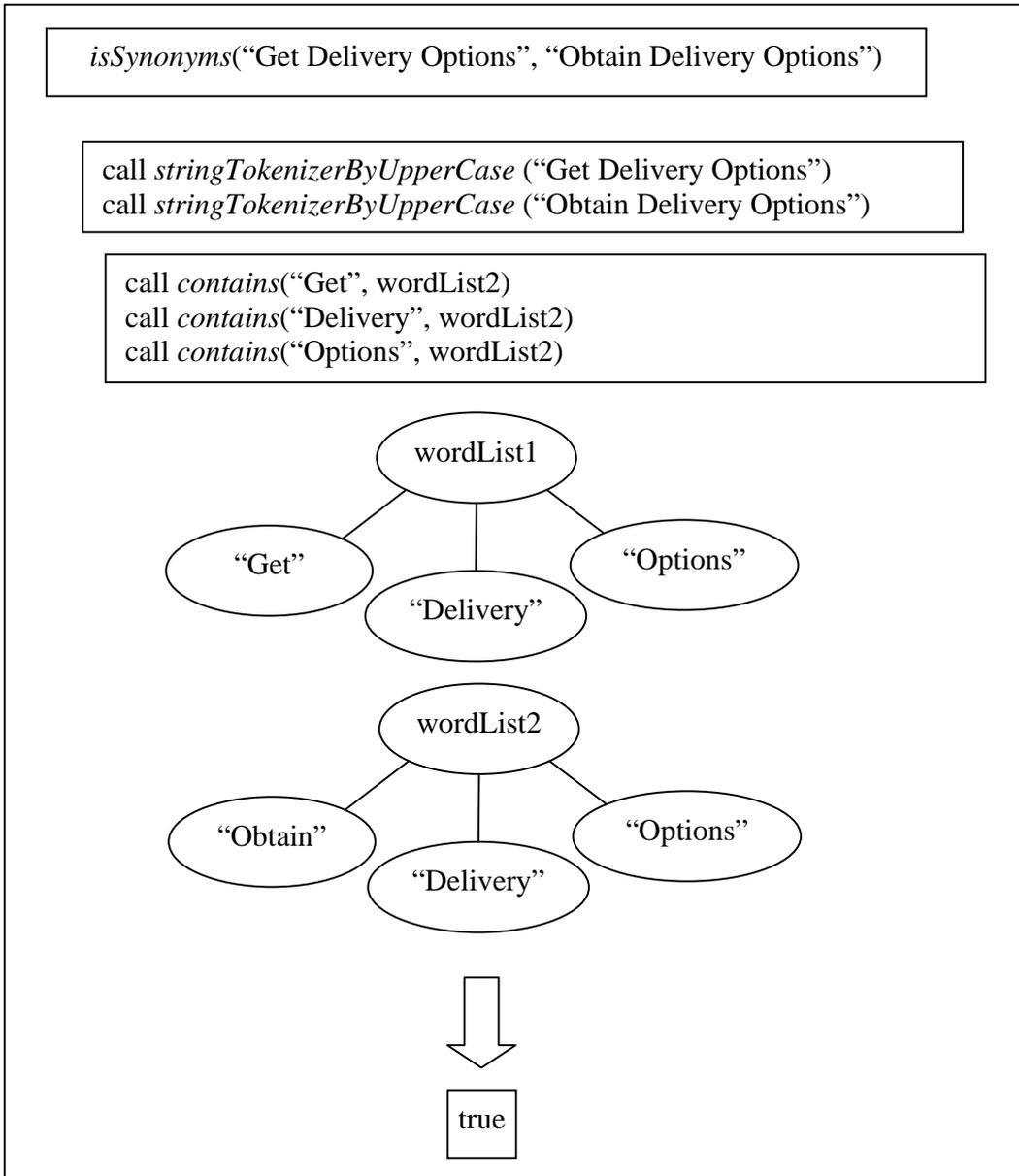


Figure A.41 `isSynonyms` function

A.1.6.4 StringTokenizer function

The *stringTokenizer* function receives as parameter a String and break down the string into tokens using as delimiters spaces, '_', '-', '(', and ')'. A list contained “Get”, “Delivery” and “Options” String is returned as result when function *stringTokenizer* function is called passing as parameter “Get Delivery Options” String.

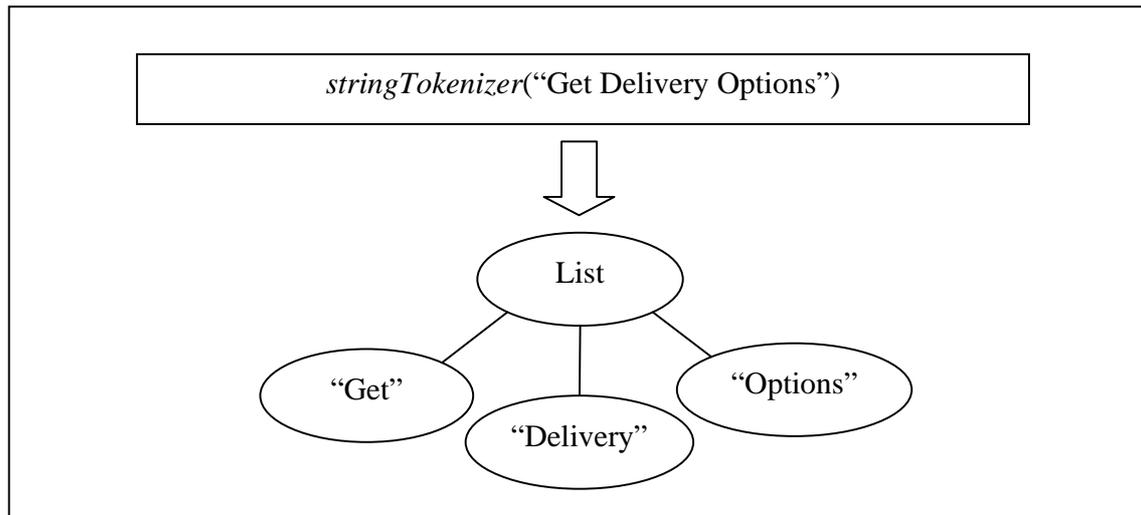


Figure A.42 stringTokenizer function example

A.1.6.5 Contains function

The contains function receives as parameter Strings *list1* and *list2* and then call contains functions (see Section A.1.6.2.) to check if elements in the *list1* are contained by the *list2*. Figure A.43 shows an example where *list1* consists of the “Get”, “Delivery” and “Options” Strings and *list2* consists of “Obtain”, “Delivery”, and “Option” Strings. The *contains* function returns *true* when it is invoked and *list1* and *list2* are passed as parameter.

isSynonyms("Get Delivery Options", "Obtain Delivery Options")

call contains("Get", wordList2)
call contains("Delivery", wordList2)
call contains("Options", wordList2)

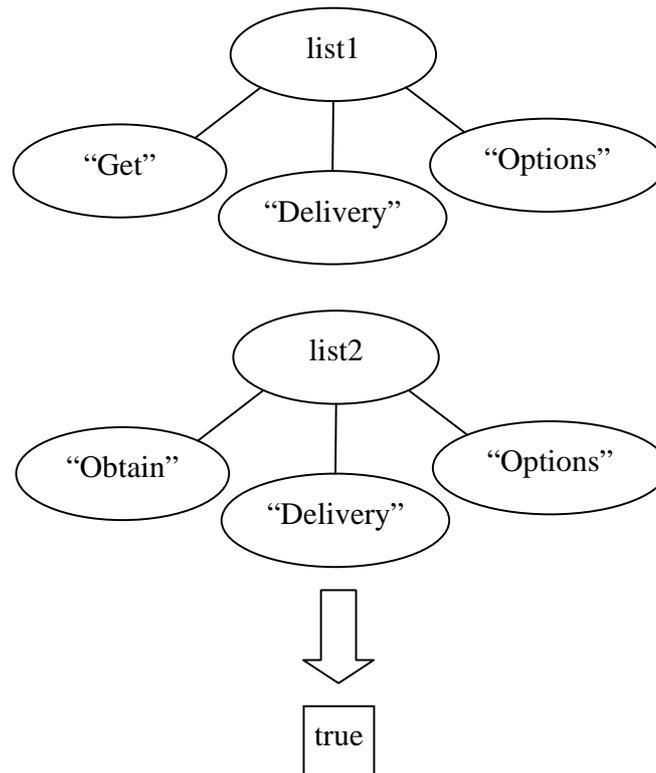


Figure A.43 *isSynonyms* function example

A.1.7 XQueryTAOMFunctions

Method Summary	
String	<u>getAttributeValue</u> (TinyNodeImpl node, String attributeName)
ArrayList< <u>TraceElement</u> >	<u>getSubElements</u> (ArrayList< <u>TraceElement</u> > subElements, String subElementId)
ArrayList< <u>TraceElement</u> >	<u>getSubElements</u> (TinyNodeImpl node)
ArrayList< <u>TraceElement</u> >	<u>getSubGoalsAndTask</u> (ArrayList< <u>TraceElement</u> > subElements, String subElementId)
ArrayList< <u>TraceElement</u> >	<u>getSubGoalsAndTask</u> (TinyNodeImpl node)
String	<u>getTAOMFileName</u> ()

Table A.7 XQueryTAOMFunctions

A.1.7.1 GetSubGoalsAndTask function

The *getSubGoalsAndTask* function receives an id of an element in i^* and returns the sub-goals and sub-tasks that are part of means-end and decomposition links. If a sub-element has sub-elements then the function calls itself recursively. For instance, Figure A.45 shows an example when the function is called to retrieve sub-elements of the Landing task (xmi:id = “_4cvccCQkEd6fbcMfsKI3Cw”) in i^* . The function returns a list of elements that consists of the Assign Slot, Initiate Approach and Follow Approach elements.

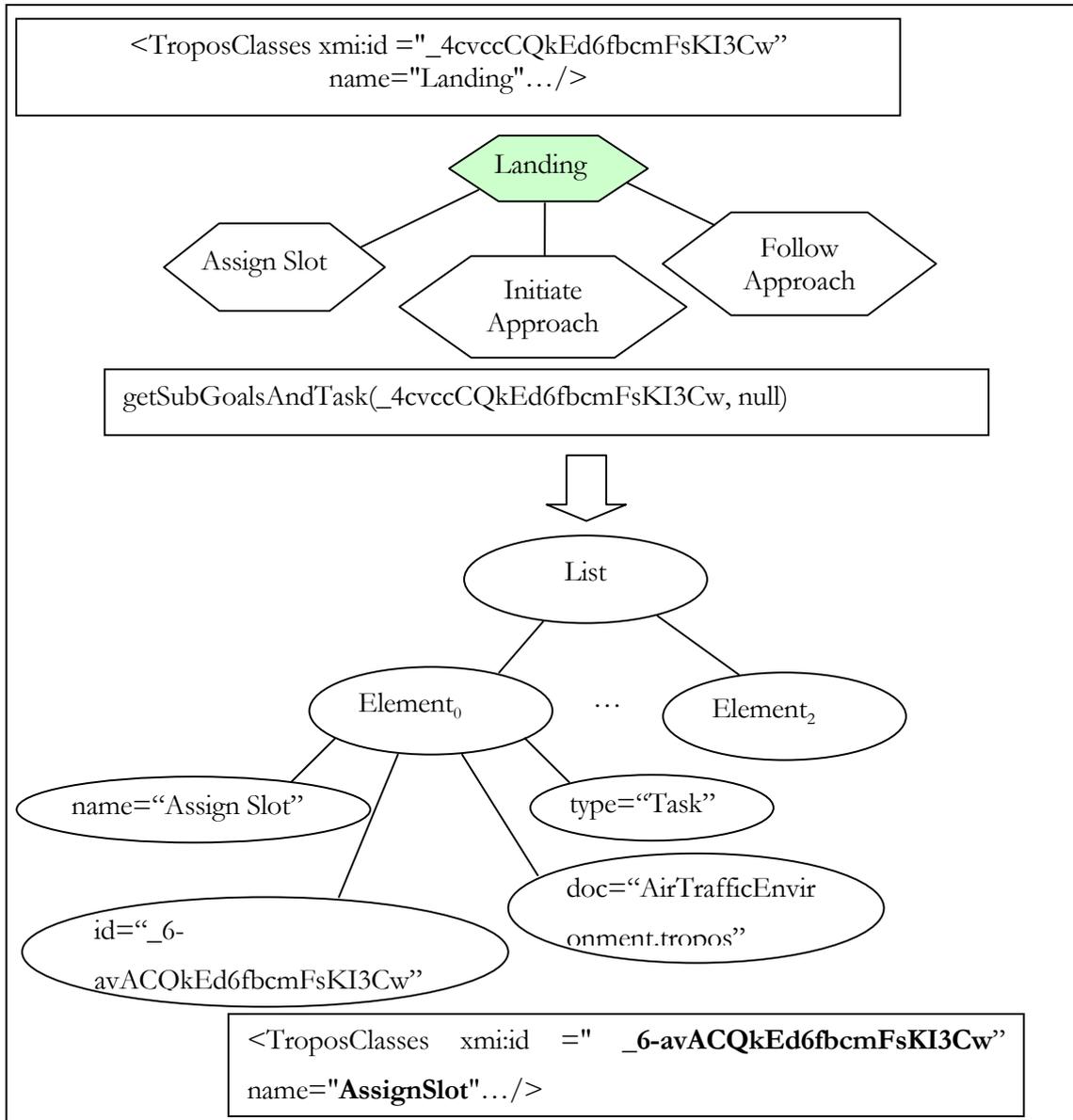


Figure A.44 getSubGoalsAndTask function example

A.1.7.2 GetSubGoalsAndTask function

The *getSubGoalsAndTask* function receives as parameter a *TinyNodeImpl* node that represents an XML node element in Saxon. The function calls *getSubGoalsAndTask* function explained in the Section 4.1.7.1 and returns a list of sub-goals and sub-tasks that are a part of means-end and decomposition links. Figure A.45 shows an example when the *getSubGoalAndTask* function is called to retrieve the sub-elements of the node that contains the Landing task in *. The

function returns a list of elements contained Assign Slot, Initiate Approach and Follow Approach.

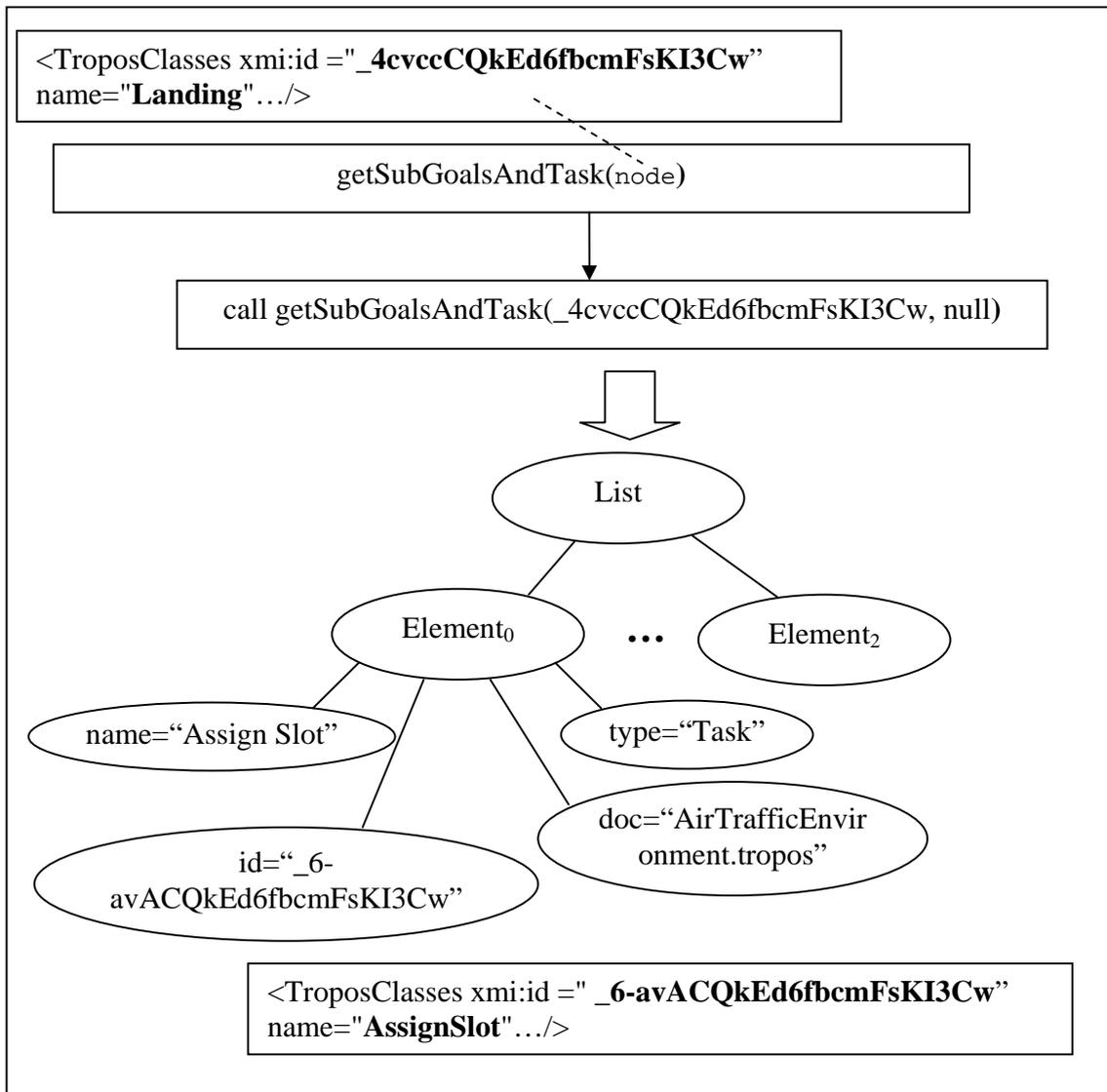


Figure A.45 getSubGoalsAndTask function example

A.1.7.3 GetSubElements function

The *getSubElements* function receives an *id* of an element in *i** and returns the sub-elements that are part of *means-end* and *decomposition* links. If a sub-element has sub-elements then the function calls itself recursively. The *getSubElements* function behaviour is similar to the *getSubGoalsAndTask* function explained in the Section 4.1.7.1., except that it returns all types of sub-elements.

A.1.7.4 GetSubElements function

The *getSubElements* function receives as parameter a *TinyNodeImpl* node that represents an XML node element in Saxon. The function calls *getSubElements* function explained in the Section A.1.7.3 and returns a list of sub-elements that are a part of *means-end* and *decomposition* links.

A.1.7.5 GetAttributeValue function

The *getAttributeValue* function returns the value of an attribute of an XML Element in Saxon. The function receives two parameters a *TinyNodeImpl* node and *String* attributeName. The node represents a XML element in the Saxon. For instance, Figure A.46 shows a *TinyNodeImpl* node in Saxon that represents the object XML element shown in the Figure A.47. If you call the *getAttributeValue* function and pass as parameter node and the String “type” then the function returns the value “Agent” as result.

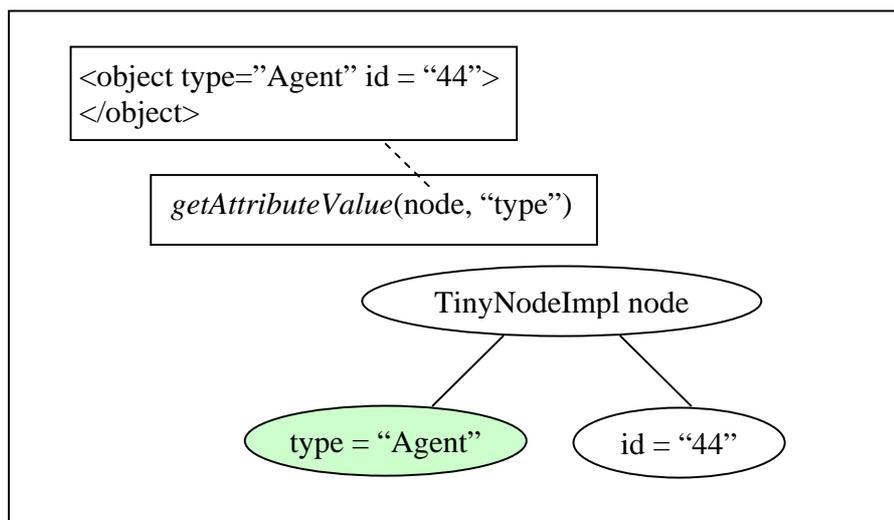


Figure A.46 *getAttributeValue* function example

A.1.7.6 GetTAOMFile function

The *getTAOMFile* functions returns the TAOM filename defined during the creation of the project and that is used to generate traceability relations between i^* elements.

Appendix B – Automated Teller Machine

B.1 Introduction

This document describes the development of a multi-agent system to implement the Automated Teller Machine (ATM) used as a case study to evaluate our approach to generate traceability relations automatically and to identify missing elements between artefacts created during the development of a multi-agent system.

Automated Teller Machines (ATMs) allow customers to carry out bank transactions without the assistance of a teller such as withdraw cash, change PIN, make a payment, check balance, print statement, and transfer money. The customer needs to insert a card in the ATM machine and enter a PIN code to use one of services provided by the ATM machine. When the customer inserts the card the system reads the card details and shows a screen asking for a PIN number. The customer enters the PIN number and then the system validates the PIN number. If the PIN number is correct the system shows a screen with the services available for the customer.

If the customer selects withdraw cash option, the system shows withdraw cash screen. The customer enters the amount of money that he/she would like to withdraw and then the system processes the cash withdraw. The system requests to the Bank authorization to the cash withdraw. If the Bank approve the cash withdraw, the ATM machine dispenses the amount of cash requested by the customer and prints a receipt. If the Bank does not approve the cash withdraw, the ATM machine shows a message given details why the cash withdraw was not authorized.

If the customer selects to change the PIN number, the ATM machine shows a screen where the customer can enters a new PIN number. After the customer enters the new PIN number, the ATM sends the new PIN number to the Bank. If the customer selects to make a payment, the ATM shows a screen where the customer can enter details about the payment. The ATM sends details about the payment to the Bank execute the payment. If the customer selects the balance account, the ATM requests the balance to Bank and then shows the balance on the screen. If the customer selects to print statement, the ATM requests the transactions done by the customers to the Bank and then prints the transactions details. If the customer selects to

transfer money, the ATM shows a screen where the customer can enter details about the account to transfer money and amount to transfer. The ATM sends the details about the money transfer to the Bank to execute the transfer.

Support Staff periodically performs maintenance on the ATM machine. Support Staff replaces tonner when receives alert of low quantity of tonner. Support Staff deposit more cash into the ATM machine when receives alert of low quantity of cash. Support Staff put in more paper for the printer when alerted of paper out. Support Staff performs maintenance when alerted of paper jam.

The remainder of this document describes the development of the ATM case study and its evaluation. The ATM case study was developed using i^* framework to model the organizational environment, Prometheus methodology to create the system specification, analysis and design models and JACK Intelligent Agent language to implement the multi-agent system. To evaluate our approach we use precision and recall measures to show the effectiveness of the traceability recovery by the approach and used the missing element information identified by the tool to complete the models and to fix inconsistencies (e.g. to fix discrepancies between names given by the elements).

B.2 Organizational Models

The Automatic Teller Machine environment is composed of the ATM, Bank, Support Staff, Power Supply, and Customer actors. Figure B.1 shows the actors and its strategic dependencies relationships. The Customer actor depends on ATM actor to have Withdraw Cash, Transfer Money, Make a Payment, Show Services Available, Check Balance, Print Statement, and Change PIN goals accomplished. The Customer depends on the ATM to have Balance, Receipt, Statement, and Cash resources provided by the ATM actor. The ATM actor depends on the Bank actor to have Provide Services, Process Withdraw, and Authorization Response goals achieved. The ATM actor depends on the Support Staff actor to have Execute Maintenance goal accomplished. The Support Staff actor depends on the ATM actor to have Alert Low Quantity of Tonner, Alert Low Quantity of Cash, Alert Paper Jam, and Alert Paper Out goals achieved.

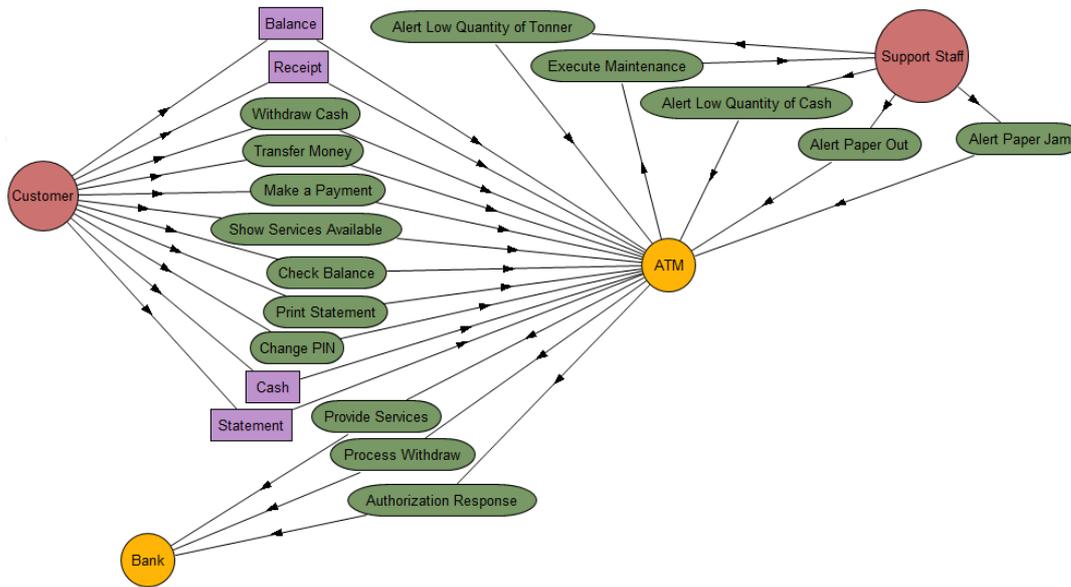


Figure B.1 Strategic Dependency model for the Automatic Teller Machine

Figure B.2 shows a partial Strategic Rationale model for the Automatic Teller Machine. Figure B.2 describes in more detail how the ATM actor achieves the Withdraw Cash goal dependency. The ATM actor performs Request Withdraw and Process Request Response tasks in order to achieve Withdraw Cash goal. The ATM actor depends on the Bank actor to send Authorization Response in order to execute Process Request Response task. If the Bank approves the cash withdraw the ATM actor performs the Approve Withdraw task and if the Bank rejects the cash withdraw the ATM performs Reject Withdraw task. The ATM actor uses Cash and Receipt resources and executes Eject Card and Show Withdraw Message Approved tasks to complete the Approve Withdraw task. The ATM actor performs Eject Card and Show Withdraw Request Rejected tasks in order to complete Reject Withdraw task. The Bank needs balances and accounts resource information to have Process Withdraw goal achieved.

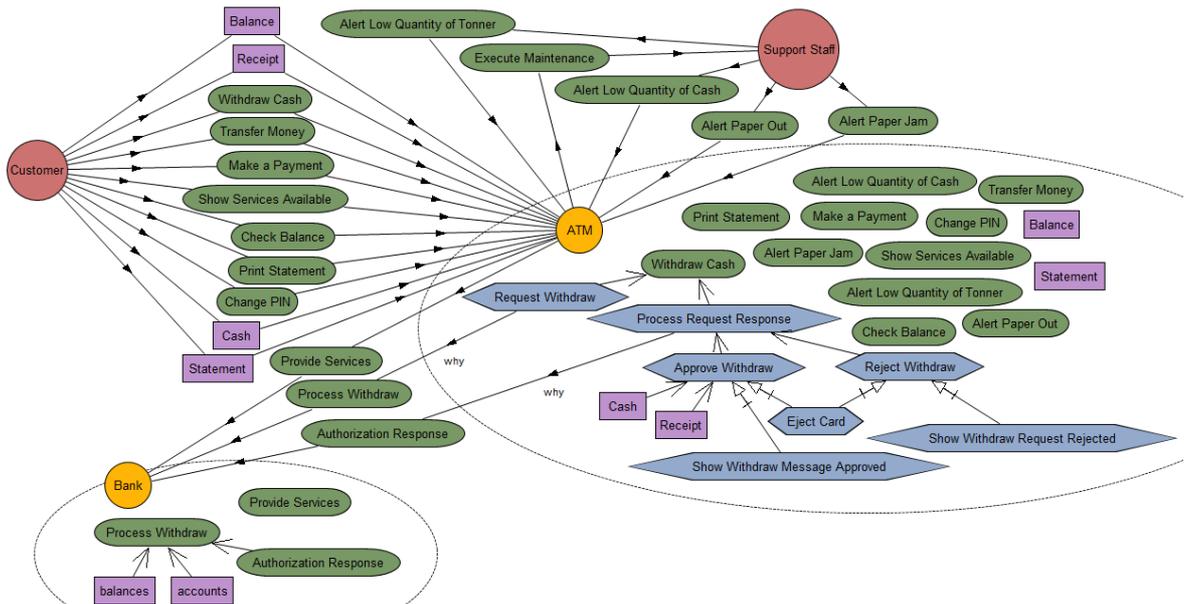


Figure B.2 Strategic Rationale Model for the Automatic Teller Machine

B.3 Prometheus Models

Figure B.3 shows the Goal diagram for the ATM system. In the top level, we have the Withdraw Money goal that is refined by the Authorize Withdraw goal that is decomposed in Request Approved or Requested Rejected.

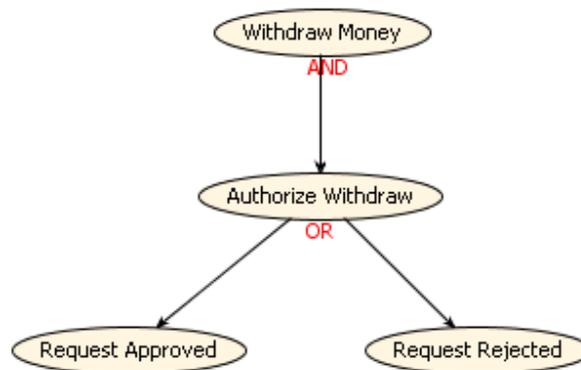


Figure B.3 ATM Goal diagram

The Automatic Teller Machine application consists of a multi-agent systems composed of two agents: Atm and Bank. Figure B.4 shows a System Overview Diagram for the Automatic Teller Machine multi-agent system. The Atm agent perceives when a customer inserts a card, requests to withdraw cash and selects to print a statement represented in diagram by the Card Inserted, Withdraw Percept, and Print Statement percepts. The Atm agent process information from the Withdraw Percept and it posts a Withdraw message. The Atm agent handles the

Withdraw message and it sends a Withdraw Request message passing amount, account and pin information details to the Bank agent. The Bank agent handles the Withdraw Request message and replies to the Atm agent with WithdrawResponse passing balance and approval/rejection information details to the Atm agent. The Atm agent executes Show Message Remove Cash, Dispense Cash, Print Receipt, and Eject Card actions if the Bank agent approved the cash withdraw and it executes Eject Card and Show Message Request Rejected actions if the Bank agent rejected the cash withdraw.

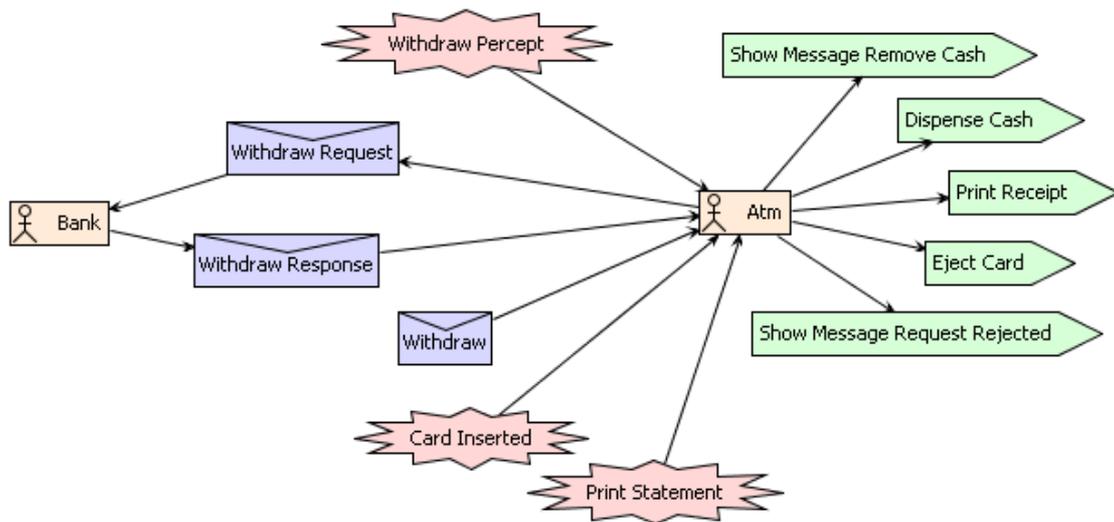


Figure B.4 ATM System Overview diagram

Figure B.5 and B.6 show the Agent Overview diagrams for the Atm and Bank agents, respectively. In the Figure B.5, the Withdraw Cash plan handles the Withdraw message and sends Withdraw Request message to the Bank agent. In the Figure B.5, the Process Withdraw plan handles Withdraw Request message, reads the account details, modifies the balance if there is sufficient funds and posts a Withdraw Response message with the new balance and the approval information or the rejection information if the withdraw has failed because insufficient funds in the account. In the Figure B.5, the Withdraw Approved and Withdraw Rejected plans handle the Withdraw Response message. If the withdraw has been approved, the Withdraw Approved plan is selected to handle the Withdraw Response message otherwise the Withdraw Rejected plan is selected. The Withdraw Approved handles the Withdraw Response message and executes Eject Card, Dispense Cash, Show Message Remove Cash, and Print Receipt actions. The Withdraw Rejected plan handles Withdraw Response message and executes the Eject Card and Show Message Request Rejected actions.

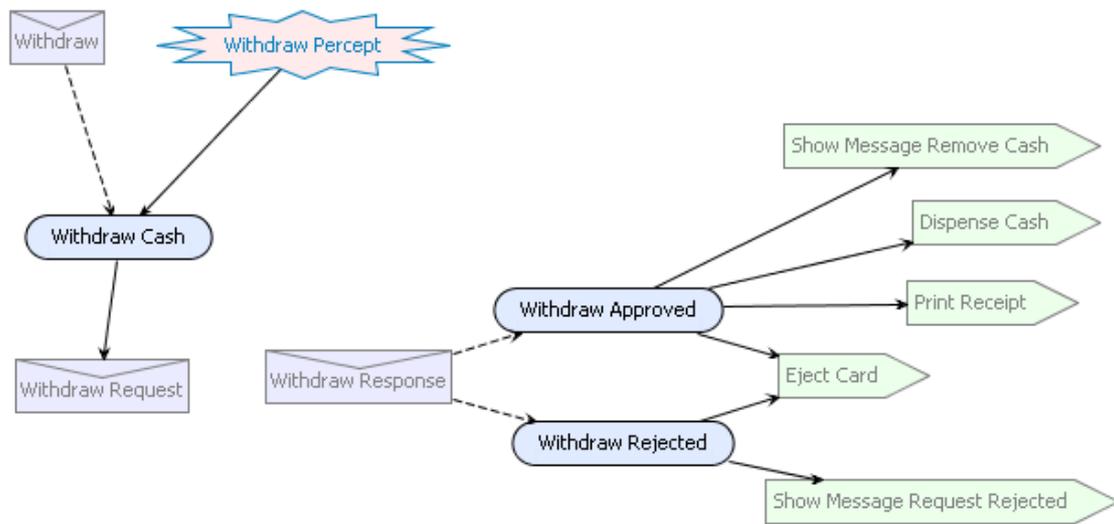


Figure B.5 – Atm Agent Overview Diagram

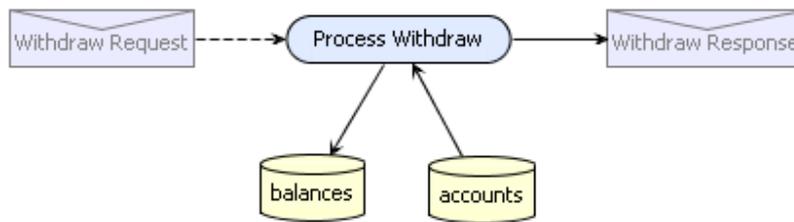


Figure B.6 Bank Agent Overview diagram

B.4 JACK Code

The multi-agent system was implemented using JACK Agent Language. The system consists of Atm and BankAgent agents, Accounts and Balances beliefsets, Withdraw, WithdrawRequest, and WithdrawResponse events, ProcessWithdraw, WithdrawApproved, WithdrawCash, and WithdrawRejected plans.

Figure B.7 shows Atm agent. The Atm agent is part of the package agent and it uses aos.jack.jak.core, gui.AtmClient, and AtmInterface packages. The Atm agent handles Withdraw and WithdrawResponse events. The Atm agent sends WithdrawRequest event and posts Withdraw events. The Atm agent uses WithdrawCash, WithdrawApproved, and WithdrawRejected plans. The Atm agent has AtmInterface, account, pin, amount, and bank attributes. The Atm agent implements getHardware, getAccount, and statement methods.

```

package agents;

import aos.jack.jak.core.*;
import gui.AtmClient;
import gui.AtmInterface;

public agent Atm extends Agent implements AtmClient {

    #handles event Withdraw;
    #handles event WithdrawResponse;
    #sends event WithdrawRequest;
    #posts event Withdraw request;

    #uses plan WithdrawCash;
    #uses plan WithdrawApproved;
    #uses plan WithdrawRejected;

    private AtmInterface hardware;
    private int account;
    private int pin;
    private int amount;
    private String bank;

    public Atm(String n, AtmInterface h, String b) {
        super(n);
        bank = b;
        hardware = h;
        h.register((AtmClient)this);
    }

    /* The AtmClient implementation */
    public AtmInterface getHardware(){
        return hardware;
    }
}

```

```

public int getAmount(){
    return amount;
}
public String getBank(){
    return bank;
}
public void insertCard(int account, int pin){
    this.account = account;
    this.pin = pin;
}
public void withdraw(int amount){
    this.amount = amount;
    postEvent(request.withdraw());
}

public void statement(){
    throw new Error("statement() is not implemented.");
}
}

```

Figure B.7 Atm agent

Figure B.8 shows BankAgent agent. The BankAgent agent is part of the agents package (package agents). The BankAgent agent handles WithdrawRequest event and uses ProcessWithdraw plan. The BankAgent agent contains accounts and balances beliefSets.

```

package agents;

public agent BankAgent extends Agent {

    #sends event WithdrawResponse response;

    #handles event WithdrawRequest;
    #uses plan ProcessWithdraw;

    #private data Accounts accounts("accounts.dat");
    #private data Balances balances("balances.dat");

    public BankAgent(String n){
        super(n);
        try {
            if (accounts.nFacts() <= 0) {
                accounts.add(10, 10);
                balances.add(10, 1000);
            }
        } catch (Exception e) {}
    }
}

```

Figure B.8 BankAgent agent

Figure B.9 shows Accounts beliefSet that contains account and pin fields and the query function query. The account field is declared as key field. The Accounts beliefSet implements addfact, newfact, endfact, delfact, modfact, modddb callback methods.

```

package agents;

public beliefset Accounts extends OpenWorld {

    #key field int account;
    #value field int pin;

    #indexed query query(int i, int j);

    public void addfact(Tuple t, BeliefState d){
        System.err.println("Accounts: addfact " + t.toString() +
            ":" + d.toString());
    }

    public void newfact(Tuple t, BeliefState d, BeliefState old){
        System.err.println("Accounts: newfact " + t.toString() +
            ":" + d.toString() + ":" + old.toString());
    }

    public void endfact(Tuple t, BeliefState old, BeliefState d){
        System.err.println("Accounts: endfact " + t.toString() +
            ":" + old.toString() + ":" + d.toString());
    }

    public void delfact(Tuple t, BeliefState d){
        System.err.println("Accounts: delfact " + t.toString() +
            ":" + d.toString());
    }

    public void modfact(Tuple t, BeliefState d, Tuple tr, Tuple fl){
        String s;
        System.err.println("Accounts: endfact " + t.toString() +
            ":" + d.toString());

        if (tr == null)
            s = "null";
        else
            s = tr.toString();
        System.err.println("\t"+ s + ":true");
        if (fl == null)
            s = "null"
        else
            s = fl.toString();
        System.err.println("\t"+ s + ":false");
    }

    public void modddb(){
        write("new_accounts.dat");
        System.err.println("Accounts: "+"modddb");
    }
}

```

Figure B.9 Accounts beliefSet

Figure B.10 shows Balances beliefSet that contains account and balance fields and the query function query. The account field is declared as key field. The Balances beliefSet implements addfact, newfact, endfact, delfact, modfact, modddb callback methods.

```

package agents;

public beliefset Balances extends OpenWorld {

    #key field int account;
    #value field int balance;

    #indexed query query(int i, logical int j);

    public void addfact(Tuple t, BeliefState d){
        System.err.println("Balances: addfact " + t.toString() +
            ":" + d.toString());
    }

    public void newfact(Tuple t, BeliefState d, BeliefState old){
        System.err.println("Balances: newfact " + t.toString() +
            ":" + d.toString() + ":" + old.toString());
    }

    public void endfact(Tuple t, BeliefState old, BeliefState d){
        System.err.println("Balances: endfact " + t.toString() +
            ":" + old.toString() + ":" + d.toString());
    }

    public void delfact(Tuple t, BeliefState d){
        System.err.println("Balances: "+"delfact " + t.toString() +
            ":" + d.toString());
    }

    public void modfact(Tuple t, BeliefState d, Tuple tr, Tuple fl){
        String s;
        System.err.println("Balances: "+"endfact " + t.toString() +
            ":" + d.toString());

        if (tr == null)
            s = "null";
        else
            s = tr.toString();
        System.err.println("\t"+ s + ":true");
        if (fl == null)
            s = "null";
        else
            s = fl.toString();
        System.err.println("\t"+ s + ":false");
    }

    public void modddb(){
        write("new_balances.dat");
        System.err.println("Balances: "+"modddb");
    }
}

```

Figure B.11 shows Withdraw event that implements withdraw posting method.

```
package agents;

import aos.jack.jak.core.*;

event Withdraw extends Event {
    #posted as
    withdraw() {
        Jak.log.log("Withdraw:withdraw created");
    }
}
```

Figure B.11 Withdraw event

Figure B.12 shows WithdrawResponse event that contains approved, and balance data members and implements approval and rejection posting methods.

```
package agents;

import aos.jack.jak.core.Jak;

public event WithdrawResponse extends MessageEvent {

    public boolean approved;
    public int balance;

    #posted as
    approval(int balance){
        Jak.log.log("WithdrawResponse:approval created");
        this.approved = true;
        this.balance = balance;
        message = "approved";
    }

    #posted as
    rejection(){
        Jak.log.log("WithdrawResponse:rejection created");
        this.approved = false;
        this.balance = 0;
        message = "rejected";
    }
}
```

Figure B.12 WithdrawResponse event

Figure B.13 shows WithdrawRequest event that contains account, pin and amount data members and implements withdraw posting method.

```

package agents;

import aos.jack.jak.core.Jak;

event WithdrawRequest extends MessageEvent {
    public int account;
    public int pin;
    public int amount;

    #posted as
    withdraw(int account, int pin, int amount) {
        Jak.log.log("WithdrawRequest:withdraw created");
        this.account = account;
        this.pin = pin;
        this.amount = amount;
        message = "withdraw["+account+", "+pin+"]";
    }
}

```

Figure B.13 WithdrawRequest event

Figure B.14 shows *WithdrawApproved* plan that handles *WithdrawApproved* event and uses *AtmClient* agent declaration. The *WithdrawApproved* implements *context* method and *body* reasoning method.

```

import gui.AtmClient;
import gui.AtmInterface;

public plan WithdrawApproved extends Plan {

    #handles event WithdrawResponse event;
    #uses agent implementing AtmClient atmclient;

    context() {
        event.approved;
    }

    #reasoning method
    body(){
        AtmInterface hardware = atmclient.getHardware();
        hardware.message("Remove your cash");
        hardware.dispense(atmclient.getAmount());
        @sleep(5.0);
        hardware.receipt(atmclient.getAccount(), event.balance);
        @sleep(5.0);
        hardware.eject();
    }
}

```

Figure B.14 WithdrawApproved plan

Figure B.15 shows *WithdrawCash* plan that handles *Withdraw* event, sends *WithdrawRequest* event and uses *AtmClient* agent declaration. The *WithdrawCash* plan implements *context* method and *body* reasoning methods.

```

package agents;

import gui.AtmInterface;
import gui.AtmClient;

public plan WithdrawCash extends Plan {
    #handles event Withdraw event;
    #uses agent implementing AtmClient atmcc;

    context() {
        atmcc.getHardware().cardInserted();
    }

    #sends event WithdrawRequest request;
    #reasoning method
    body(){
        @send(atmcc.getBank(), request.withdraw( atmcc.getAccount(),
            atmcc.getPin(), atmcc.getAmount()
            ));
    }
}

```

Figure B.15 WithdrawCash plan

Figure B.16 shows *WithdrawRejected* plan that handles *WithdrawResponse* event and uses *AtmClient* agent declaration. The *WithdrawRejected* plan implements *relevant* method and *body* reasoning method.

```

package agents;

import gui.AtmInterface;
import gui.AtmClient;

public plan WithdrawRejected extends Plan {
    #handles event WithdrawResponse event;
    #uses agent implementing AtmClient atmcc;

    static boolean relevant(WithdrawResponse event){
        return !event.approved;
    }
    #reasoning method
    body(){
        AtmInterface hardware = atmcc.getHardware();
        hardware.message("Request rejected");
        @sleep(5.0);
        hardware.eject();
        atmcc.getHardware().eject();}}

```

Figure B.16 WithdrawRejected plan

Figure B.17 shows ProcessWithdraw plan that handles WithdrawRequest event, posts WithdrawResponse event, reads accounts beliefset, and modifies balances beliefset. The ProcessWithdraw implements body and fail reasoning methods.

```

package agents;
public plan ProcessWithdraw extends Plan {
    #handles event WithdrawRequest event;

    #posts event WithdrawResponse response;
    #reads data Accounts accounts;
    #modifies data Balances balances;

    #reasoning method
    body(){
        logical int balance;
        int new_balance;
        (event.amount >= 0);
        try {
            if (!accounts.query(event.account,event.pin))
                System.err.println("accouts query failed");
            (accounts.query(event.account,event.pin) &&
             balances.query(event.account,balance));
            (event.amount <= balance.getValue());
            new_balance = (balance.getValue() - event.amount);
            balances.add(event.account,new_balance);
            @send(event.from,response.approval(new_balance));
        }
        catch (Exception e) {
            Jak.log.log(("ProcessWithdraw caught exception" + e));
            e.printStackTrace();
            (false == true);
        }
    }
    #reasoning method
    fail(){
        @send(event.from,response.rejection());
    }
}

```

Figure B.17 WithdrawApproved plan

B.5 JACK Code in XML

This section show the JACK code presented in the Section 4 converted in XML format. The Figure B.18 shows Atm agent in XML.

```

<agent id="ag1" name="Atm" extends="Agent" implements="AtmClient">
  <import>gui.AtmClient</import>
  <import>aos.jack.jak.core.*</import>
  <import>gui.AtmClient</import>
  <import>gui.AtmInterface</import>
  <handlesEvent>Withdraw</handlesEvent>
  <handlesEvent>WithdrawResponse</handlesEvent>

```

```

<sendsEvent type="WithdrawRequest"/>
<postsEvent type="Withdraw" ref="request"/>
<usesPlan> WithdrawCash</usesPlan>
<usesPlan>WithdrawApporoved</usesPlan>
<usesPlan>WithdrawRejected</usesPlan>
<attribute type="AtmInterface" ref="hardware"/>
<attribute type="int" ref="account"/>
<attribute type="int" ref="pin"/>
<attribute type="int" ref="amount"/>
<attribute type="String" ref="bank"/>
<constructor>
  <parameter type="String" ref="n"/>
  <parameter type="AtmInterface" ref="h"/>
  <parameter type="String" ref="b"/>
  <body>
    <![CDATA[
      super(n);
      bank = b;
      hardware = h;
      h.register((AtmClient)this);
    ]]>
  </body>
</constructor>
<method name="getHardware" returnType="AtmInterface">
  <body>
    <![CDATA[
      return hardware;
    ]]>
  </body>
</method>
<method name="getAccount" returnType="int">
  <body>
    <![CDATA[
      return account;
    ]]>
  </body>
</method>
<method name="getPin" returnType="int">
  <body>
    <![CDATA[
      return pin;
    ]]>
  </body>
</method>
<method name="getAmount" returnType="int">
  <body>
    <![CDATA[
      return amount;
    ]]>
  </body>
</method>
<method name="getBank" returnType="String">
  <body>
    <![CDATA[
      return bank;
    ]]>
  </body>
</method>

```

```

<method name="insertCard" returnType="void">
  <parameter type="int" ref="account"/>
  <parameter type="int" ref="pin"/>
  <body>
    <![CDATA[
      this.account = account;
      this.pin = pin;
    ]]>
  </body>
</method>
<method name="withdraw" returnType="void">
  <parameter type="int" ref="amount"/>
  <body>
    <![CDATA[
      this.amount = amount;
      postEvent(request.withdraw());
    ]]>
  </body>
</method>
<method name="statement" returnType="void">
  <parameter type="int" ref="amount"/>
  <body>
    <![CDATA[
      throw new Error("statement() is not implemented.");
    ]]>
  </body>
</method>
</agent>

```

Figure B.18 Atm agent in XML

Figure B.19 shows *BankAgent* agent in XML.

```

<agent id="ag2" name="BankAgent" extends="Agent" >
  <sendsEvent type="WithdrawResponse" ref="response" />
  <handlesEvent>WithdrawRequest</handlesEvent>
  <privateData beliefType="Accounts" belieName="accounts"/>
  <privateData beliefType="Balances" belieName="balances"/>
  <constructor>
    <parameter type="String" ref="n"/>
    <body>
      <![CDATA[
        super(n);
        try {
          if (accounts.nFacts() <= 0) {
            accounts.add(10, 10);
            balances.add(10, 1000);
          }
        } catch (Exception e) {}
      ]]>
    </body>
  </constructor>
</agent>

```

Figure B.19 BankAgent in XML

Figure B.20 shows *Accounts* beliefSet.

```
<beliefSet id="b1" type="Accounts" extends="OpenWorld">
  <field declarationType="key" type="int" name="account"/>
  <field declarationType="value" type="int" name="pin"/>
  <indexedQuery methodName="query">
    <parameters>
      <parameter type="int" members="normal" ref="i"/>
      <parameter type="int" members="normal" ref="j"/>
    </parameters>
  </indexedQuery>
  <method name="addfact" returnType="void">
    <parameter type="Tuple" ref="t"/>
    <parameter type="BeliefState" ref="d"/>
    <parameter type="BeliefState" ref="was"/>
    <body>
      <![CDATA[
        System.err.println("Accounts: addfact " + t.toString() +
          ":" + d.toString());
      ]]>
    </body>
  </method>
  <method name="newfact" returnType="void">
    <parameter type="Tuple" ref="t"/>
    <parameter type="BeliefState" ref="d"/>
    <parameter type="BeliefState" ref="old"/>
    <body>
      <![CDATA[
        System.err.println("Accounts: newfact " + t.toString()
          + ":" + d.toString() + ":" + old.toString());
      ]]>
    </body>
  </method>
  <method name="endfact" returnType="void">
    <parameter type="Tuple" ref="t"/>
    <parameter type="BeliefState" ref="old"/>
    <parameter type="BeliefState" ref="d"/>
    <body>
      <![CDATA[
        System.err.println("Accounts: endfact " + t.toString() +
          ":" + old.toString() + ":" + d.toString());
      ]]>
    </body>
  </method>
  <method name="delfact" returnType="void">
    <parameter type="Tuple" ref="t"/>
    <parameter type="BeliefState" ref="d"/>
    <body>
      <![CDATA[
        System.err.println("Accounts: delfact " + t.toString() +
          ":" + d.toString());
      ]]>
    </body>
  </method>
  <method name="modfact" returnType="void">
    <parameter type="Tuple" ref="t"/>
    <parameter type="BeliefState" ref="d"/>
    <parameter type="Tuple" ref="tr"/>
    <parameter type="Tuple" ref="fl"/>
  </method>
</beliefSet>
```

```

<body>
  <![CDATA[
    String s;
    System.err.println("Accounts: endfact " + t.toString() +
      ":" + d.toString());
    if (tr == null)
      s = "null";
    else
      s = tr.toString();
    System.err.println("\t" + s + ":true");
    if (fl == null)
      s = "null";
    else
      s = fl.toString();
    System.err.println("\t" + s + ":false");
  ]]>
</body>
</method>
<method name="modddb" returnType="void">
  <body>
    <![CDATA[
      write("new_accounts.dat");
      System.err.println("Accounts:"+modddb);
    ]]>
  </body>
</method>
</beliefSet>

```

Figure B.20 Accounts beliefSet in XML

Figure B.21 shows *Balances* beliefSet in XML.

```

<beliefSet id="b2" type="Balances" extends="OpenWorld">
  <field declarationType="key" type="int" name="account"/>
  <field declarationType="value" type="int" name="balance"/>
  <indexedQuery methodName="query">
    <parameters>
      <parameter type="int" members="normal" ref="i"/>
      <parameter type="int" members="logical" ref="j"/>
    </parameters>
  </indexedQuery>
  <method name="addfact" returnType="void">
    <parameter type="Tuple" ref="t"/>
    <parameter type="BeliefState" ref="d"/>
    <body>
      <![CDATA[
        System.err.println("Balances: addfact " + t.toString() +
          ":" + d.toString());
      ]]>
    </body>
  </method>
  <method name="newfact" returnType="void">
    <parameter type="Tuple" ref="t"/>
    <parameter type="BeliefState" ref="d"/>
    <parameter type="BeliefState" ref="old"/>
    <body>
      <![CDATA[
        System.err.println("Balances: newfact " + t.toString() +
          ":" + d.toString() + ":" + old.toString());
      ]]>
    </body>
  </method>

```

```

<method name="endfact" returnType="void">
  <parameter type="Tuple" ref="t"/>
  <parameter type="BeliefState" ref="old"/>
  <parameter type="BeliefState" ref="d"/>
  <body>
    <![CDATA[
      System.err.println("Balances: endfact " + t.toString() +
        ":" + old.toString() + ":" + d.toString());
    ]]>
  </body>
</method>
<method name="delfact" returnType="void">
  <parameter type="Tuple" ref="t"/>
  <parameter type="BeliefState" ref="d"/>
  <body>
    <![CDATA[
      System.err.println("Balances:"+delfact " + t.toString() +
        ":" + d.toString());
    ]]>
  </body>
</method>
<method name="modfact" returnType="void">
  <parameter type="Tuple" ref="t"/>
  <parameter type="BeliefState" ref="d"/>
  <parameter type="Tuple" ref="tr"/>
  <parameter type="Tuple" ref="fl"/>
  <body>
    <![CDATA[
      String s;
      System.err.println("Balances:"+endfact " + t.toString() +
        ":" + d.toString());
      if (tr == null)
        s = "null";
      else
        s = tr.toString();
      System.err.println("\t"+ s + ":true");
      if (fl == null)
        s = "null";
      else
        s = fl.toString();
      System.err.println("\t"+ s + ":false");
    ]]>
  </body>
</method>
<method name="moddb" returnType="void">
  <body>
    <![CDATA[
      write("new_balances.dat");
      System.err.println("Balances: "+"moddb");
    ]]>
  </body>
</method>
</beliefSet>

```

Figure B.21 Balances beliefSet in XML

Figure B.22 shows *ProcessWithdraw* plan in XML.

```

<plan id="p1" name="ProcessWithdraw" extends="Plan">
  <handlesEvent type="WithdrawRequest" ref="event"/>
  <postsEvent type="WithdrawResponse" ref="response"/>
  <readsData type="Accounts" ref="accounts"/>
  <modifiesData type="Balances" ref="balances"/>
  <body>
    <![CDATA[
      logical int balance;
      int new_balance;
      (event.amount >= 0);
      try {
        if (!accounts.query(event.account,event.pin))
          System.err.println("accouts query failed");
        (accounts.query(event.account,event.pin) &&
          balances.query(event.account,balance));
        (event.amount <= balance.getValue());
        new_balance = (balance.getValue() - event.amount);
        balances.add(event.account,new_balance);
        @send(event.from,response.approval(new_balance));
      }
      catch (Exception e) {
        Jak.log.log(("ProcessWithdraw caught exception" + e));
        e.printStackTrace();
        (false == true);
      }
    ]]>
  </body>
  <fail>
    <![CDATA[
      @send(event.from,response.rejection());
    ]]>
  </fail>
</plan>

```

Figure B.22 ProcessWithdraw plan in XML

```

<plan id="p2" name="WithdrawApproved" extends="Plan">
  <handlesEvent type="WithdrawResponse" ref="event"/>
  <usesAgent type="AtmClient" ref="atmc"/>
  <context>
    <![CDATA[
      event.approved;
    ]]>
  </context>
  <body>
    <![CDATA[
      AtmInterface hardware = atmc.getHardware();
      hardware.message("Remove your cash");
      hardware.dispense(atmc.getAmount());
      @sleep(5.0);
      hardware.receipt(atmc.getAccount(),event.balance);
      @sleep(5.0);
      hardware.eject();
    ]]>
  </body>
</plan>

```

Figure B.23 WithdrawApproved plan in XML

```

<plan id="p3" name="WithdrawCash" extends="Plan">
  <import>gui.AtmClient</import>
  <import>gui.AtmInterface</import>
  <handlesEvent type="Withdraw" ref="event"/>
  <sendsEvent type="WithdrawRequest" ref="request"/>;
  <usesAgent type="AtmClient" ref="atmc"/>
  <context>
    <![CDATA[
      atmc.getHardware().cardInserted();
    ]]>
  </context>
  <body>
    <![CDATA[
      @send( atmc.getBank(),
        request.withdraw( atmc.getAccount(),
                          atmc.getPin(),atmc.getAmount()));
    ]]>
  </body>
</plan>

```

Figure B.24 WithdrawCash plan in XML

Figure B.25 shows *WithdrawRejected* plan in XML.

```

<plan id="p4" name="WithdrawRejected" extends="Plan">
  <import>gui.AtmClient</import>
  <import>gui.AtmInterface</import>
  <handlesEvent type="WithdrawResponse" ref="event"/>
  <usesAgent type="AtmClient" ref="atmc"/>
  <relevant>
    <![CDATA[
      atmc.getHardware().cardInserted();
    ]]>
  </relevant>
  <context>
    <![CDATA[
      return !event.approved;
    ]]>
  </context>
  <body>
    <![CDATA[
      @send( atmc.getBank(),
        request.withdraw( atmc.getAccount(), atmc.getPin(),
                          atmc.getAmount()));
    ]]>
  </body>
</plan>

```

Figure B.25 WithdrawRejected plan in XML

Figure B.26 shows *Withdraw* event in XML.

```

<event id="ev1" type="Withdraw" extends="Event">
  <import>aos.jack.jak.core.*</import>
  <posted methodName="withdraw">
    <![CDATA[
      Jak.log.log("Withdraw:withdraw created");
    ]]>
  </posted>
</event>
<event id="ev2" type="WithdrawRequest" extends="MessageEvent">
  <import>aos.jack.jak.core.Jak</import>
  <field visibility="public" type="account" name="account"/>
  <field visibility="public" type="int" name="pin"/>
  <field visibility="public" type="int" name="amount"/>
  <posted methodName="withdraw">
    <parameter type="int" name="account"/>
    <parameter type="int" name="pin"/>
    <parameter type="int" name="amount"/>
    <![CDATA[
      Jak.log.log("WithdrawRequest:withdraw created");
      this.account = account;
      this.pin = pin;
      this.amount = amount;
      message = "withdraw["+account+", "+pin+"]";
    ]]>
  </posted>
</event>

```

Figure B.26 Withdraw event in XML

Figure B.27 shows *WithdrawResponse* event in XML

```

<event id="ev3" type="WithdrawResponse" extends="Event">
  <import>aos.jack.jak.core.Jak</import>
  <field visibility="public" type="boolean" name="approved"/>
  <field visibility="public" type="int" name="balance"/>
  <posted methodName="approval">
    <![CDATA[
      Jak.log.log("WithdrawResponse:approval created");
      this.approved = true;
      this.balance = balance;
      message = "approved";
    ]]>
  </posted>
  <posted methodName="rejection">
    <![CDATA[
      Jak.log.log("WithdrawResponse:rejection created");
      this.approved = false;
      this.balance = 0;
      message = "rejected";
    ]]>
  </posted>
</event>

```

Figure B.27 WithdrawResponse in XML

B.6 Evaluation

To evaluate our approach we identified traceability relations manually (see Table B.1) and compared the results with traceability relations identified by the tool (see Table B.2). 31 correct traceability relations had been identified by the tool and 33 traceability relations were missing. The precision and recall calculated were 100% and 48,43%, respectively.

Type	Prometheus Message	JACK Event
overlaps	Withdraw Request	WithdrawRequest
overlaps	Withdraw Response	WithdrawResponse
overlaps	Withdraw	Withdraw
Type	Prometheus Data	JACK BeliefSet
overlaps	Balances	Balances
overlaps	Account	Accounts
Type	Prometheus Plan	JACK Plan
overlaps	Process Withdraw	ProcessWithdraw
overlaps	Withdraw Approved	WithdrawApproved
overlaps	Withdraw Rejected	WithdrawRejected
overlaps	Withdraw Cash	WithdrawCash
Type	Prometheus Agent	JACK Agent
overlaps	Bank	BankAgent
overlaps	Atm	Atm
Type	Prometheus Goal	JACK Agent
achieves	Authorize Withdraw	BankAgent
achieves	Request Rejected	Atm
achieves	Withdraw Money	Atm
achieves	Request Approved	Atm
Type	Prometheus Plan	JACK Agent
uses	Process Withdraw	BankAgent
uses	Withdraw Approved	Atm
uses	Withdraw Rejected	Atm
uses	Withdraw Cash	Atm
Type	Prometheus Percept	JACK Agent
uses	Withdraw Percept	Atm
uses	Card Inserted	Atm
uses	Print Statement	Atm
Type	Prometheus Action	JACK Agent
uses	Show Message Request Rejected	Atm
uses	Eject Card	Atm
uses	Print Receipt	Atm
uses	Dispense Cash	Atm
uses	Show Message Remove Cash	Atm
Type	Prometheus Message	JACK Agent
receives	Withdraw Request	BankAgent
receives	Withdraw Response	Atm
receives	Withdraw	Atm
sends	Withdraw Response	BankAgent
sends	Withdraw Request	Atm
Type	Prometheus Goal	JACK Plan
achieves	Authorize Withdraw	ProcessWithdraw
achieves	Request Approved	WithdrawApproved
achieves	Request Rejected	WithdrawRejected
achieves	Withdraw Money	WithdrawCash
Type	Prometheus Agent	JACK Plan
uses	Bank	ProcessWithdraw
uses	Atm	WithdrawApproved

uses	Atm	WithdrawRejected
uses	Atm	WithdrawCash
Type	Prometheus Percept	JACK Plan
uses	Withdraw Percept	WithdrawCash
Type	Prometheus Action	JACK Plan
creates	Eject Card	WithdrawApproved
creates	Print Receipt	WithdrawApproved
creates	Dispense Cash	WithdrawApproved
creates	Show Message Remove Cash	WithdrawApproved
creates	Show Message Request Rejected	WithdrawRejected
creates	Eject Card	WithdrawRejected
Type	Prometheus Message	JACK Plan
sends	Withdraw Response	ProcessWithdraw
sends	Withdraw Request	WithdrawCash
receives	Withdraw Request	ProcessWithdraw
receives	Withdraw Response	WithdrawApproved
receives	Withdraw Response	WithdrawRejected
receives	Withdraw	WithdrawCash
Type	Prometheus Data	JACK Plan
uses	accounts	ProcessWithdraw
creates	balances	ProcessWithdraw
Type	Prometheus Plan	JACK BeliefSet
uses	Process Withdraw	Accounts
creates	Process Withdraw	Balances
Type	Prometheus Agent	JACK Event
sends	Atm	WithdrawRequest
sends	Bank	WithdrawResponse
receives	Bank	WithdrawRequest
receives	Atm	WithdrawResponse
receives	Atm	Withdraw
Type	Prometheus Plan	JACK Event
sends	Withdraw Cash	WithdrawRequest
sends	Process Withdraw	WithdrawResponse

Table B.1 Traceability relations identified manually

Rule ID	Type	Prometheus Message	JACK Event
rulePJ1a	overlaps	Withdraw Request	WithdrawRequest
rulePJ1a	overlaps	Withdraw Response	WithdrawResponse
rulePJ1a	overlaps	Withdraw	Withdraw
Rule ID	Type	Prometheus Data	JACK BeliefSet
rulePJ2a	overlaps	Balances	Balances
rulePJ2a	overlaps	Account	Accounts
Rule ID	Type	Prometheus Plan	JACK Plan
rulePJ3a	overlaps	Process Withdraw	ProcessWithdraw
rulePJ3a	overlaps	Withdraw Approved	WithdrawApproved
rulePJ3a	overlaps	Withdraw Rejected	WithdrawRejected
rulePJ3a	overlaps	Withdraw Cash	WithdrawCash
Rule ID	Type	Prometheus Agent	JACK Agent
rulePJ4a	overlaps	Bank	BankAgent
rulePJ4a	overlaps	Atm	Atm
Rule ID	Type	Prometheus Goal	JACK Agent
rulePJ5a	achieves	Authorize Withdraw	BankAgent
rulePJ5a	achieves	Request Rejected	Atm
rulePJ5a	achieves	Withdraw Money	Atm
rulePJ5a	achieves	Request Approved	Atm
Rule ID	Type	Prometheus Plan	JACK Agent
rulePJ9a	uses	Process Withdraw	BankAgent

rulePJ9a	uses	Withdraw Approved	Atm
rulePJ9a	uses	Withdraw Rejected	Atm
rulePJ9a	uses	Withdraw Cash	Atm
Rule ID	Type	Prometheus Percept	JACK Agent
rulePJ10a	uses	Withdraw Percept	Atm
rulePJ10a	uses	Card Inserted	Atm
rulePJ10a	uses	Print Statement	Atm
Rule ID	Type	Prometheus Action	JACK Agent
rulePJ11a	uses	Show Message Request Rejected	Atm
rulePJ11a	uses	Eject Card	Atm
rulePJ11a	uses	Print Receipt	Atm
rulePJ11a	uses	Dispense Cash	Atm
rulePJ11a	uses	Show Message Remove Cash	Atm
Rule ID	Type	Prometheus Message	JACK Agent
rulePJ12a	receives	Withdraw Request	BankAgent
rulePJ12a	receives	Withdraw Response	Atm
rulePJ12a	receives	Withdraw	Atm
rulePJ12b	sends	Withdraw Response	BankAgent
rulePJ12b	sends	Withdraw Request	Atm
Rule ID	Type	Prometheus Goal	JACK Plan
rulePJ14a	achieves	Authorize Withdraw	ProcessWithdraw
rulePJ14a	achieves	Request Approved	WithdrawApproved
rulePJ14a	achieves	Request Rejected	WithdrawRejected
rulePJ14a	achieves	Withdraw Money	WithdrawCash
Rule ID	Type	Prometheus Agent	JACK Plan
rulePJ15a	uses	Bank	ProcessWithdraw
rulePJ15a	uses	Atm	WithdrawApproved
rulePJ15a	uses	Atm	WithdrawRejected
rulePJ15a	uses	Atm	WithdrawCash
Rule ID	Type	Prometheus Percept	JACK Plan
rulePJ17a	uses	Withdraw Percept	WithdrawCash
Rule ID	Type	Prometheus Action	JACK Plan
rulePJ18a	creates	Eject Card	WithdrawApproved
rulePJ18a	creates	Print Receipt	WithdrawApproved
rulePJ18a	creates	Dispense Cash	WithdrawApproved
rulePJ18a	creates	Show Message Remove Cash	WithdrawApproved
rulePJ18a	creates	Show Message Request Rejected	WithdrawRejected
rulePJ18a	creates	Eject Card	WithdrawRejected
Rule ID	Type	Prometheus Message	JACK Plan
rulePJ19a	sends	Withdraw Response	ProcessWithdraw
rulePJ19a	sends	Withdraw Request	WithdrawCash
rulePJ19c	receives	Withdraw Request	ProcessWithdraw
rulePJ19c	receives	Withdraw Response	WithdrawApproved
rulePJ19c	receives	Withdraw Response	WithdrawRejected
rulePJ19c	receives	Withdraw	WithdrawCash
Rule ID	Type	Prometheus Data	JACK Plan
rulePJ20a	uses	accounts	ProcessWithdraw
rulePJ20b	creates	balances	ProcessWithdraw
Rule ID	Type	Prometheus Plan	JACK BeliefSet
rulePJ24a	uses	Process Withdraw	Accounts
rulePJ24b	creates	Process Withdraw	Balances
Rule ID	Type	Prometheus Agent	JACK Event
rulePJ31a	sends	Atm	WithdrawRequest
rulePJ31a	sends	Bank	WithdrawResponse
rulePJ31b	receives	Bank	WithdrawRequest
rulePJ31b	receives	Atm	WithdrawResponse
rulePJ31b	receives	Atm	Withdraw
Rule ID	Type	Prometheus Plan	JACK Event
rulePJ33a	sends	Withdraw Cash	WithdrawRequest

rulePJ33a	sends	Process Withdraw	WithdrawResponse
-----------	-------	------------------	------------------

Table B.2 Traceability relations identified by the tool

To show how missing elements identified by the tool can assist in the software development process we used the information of missing elements (Table B.3) to complete the models and to fix inconsistencies (e.g. to fix discrepancies between names given by the elements). The completeness checking rules showed that they were missing relations between JACK BeliefSet and Prometheus Data, JACK Plan and Prometheus Plan, Prometheus Plan and JACK Plan, Prometheus Goal and JACK Agent.

Rule ID	JACK BeliefSet	Prometheus Data
RulePJ2cc1	Accounts	
RulePJ2cc1	Balances	
Rule ID	JACK Plan	Prometheus Plan
RulePJ3cc1	ProcessWithdraw	
RulePJ3cc1	WithdrawApproved	
RulePJ3cc1	WithdrawCash	
RulePJ3cc1	WithdrawRejected	
Rule ID	Prometheus Plan	JACK Plan
RulePJ3cc2	Withdraw Approved	
RulePJ3cc2	Process Withdraw	
RulePJ3cc2	Withdraw Cash	
RulePJ3cc2	Withdraw Rejected	
Rule ID	Prometheus Goal	JACK Agent
RulePJ5cc1	Request Approved	
RulePJ5cc1	Request Rejected	
RulePJ5cc1	Withdraw Money	
RulePJ5cc1	Authorize Withdraw	
Rule ID	Prometheus Message	JACK Agent
RulePJ12cc1	Withdraw	

Table B.3 Missing Information

Table B.3 shows that the *RuleP2cc1* rule identified that is missing a traceability relations between *Accounts* and *Balances* beliefSets in JACK and some data in Prometheus. We look what data in Prometheus could be related to *Accounts* beliefSet and conclude that *Accounts* beliefSet in JACK should be related to *Accounts* data in Prometheus. BeliefSets in JACK and data in Prometheus are related when the name of the beliefSet and the name of data are synonyms and *included fields/aspects* properties of the data is similar to the *fields* in the beliefSet. Figure B.28 shows that *Accounts* beliefSet has *account* and *pin* fields and no *included fields/aspects* properties has been defined to the *Accounts* data (see Figure B.29).

```

<beliefSet id="b1" type="Accounts" extends="OpenWorld">
    <field declarationType="key" type="int" name="account"/>
    <field declarationType="value" type="int" name="pin"/>
    ...
</beliefSet>

```

Figure B.29 Fields of the Accounts beliefSet

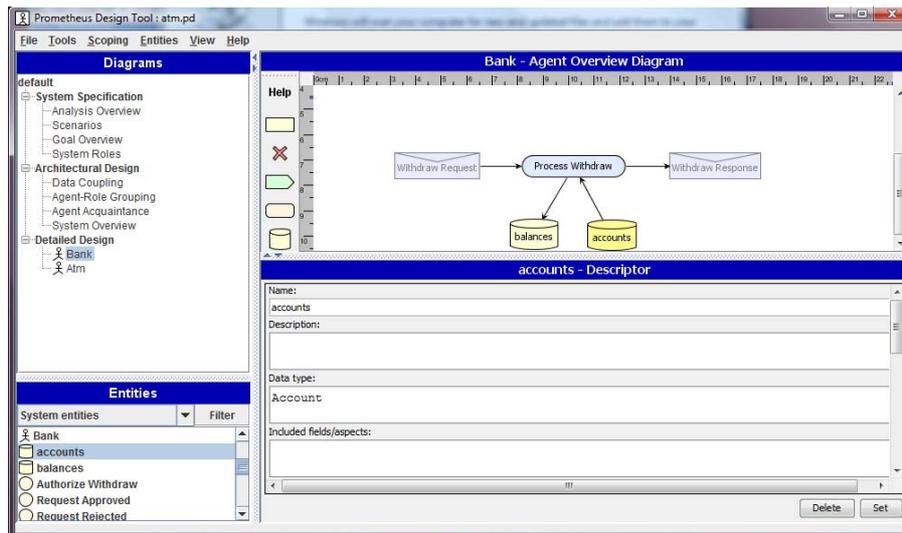


Figure B.30 Accounts beliefSet

The same way, we look what data in Prometheus could be related to *Balances* beliefSet and we concluded that *Balances* beliefSet in JACK should be related to *Balances* data in Prometheus. Figure B.31 shows that *Balances* beliefSet has *account* and *balance* fields and no *included fields/aspects* properties has been defined to the *Balances* data (see Figure B.32).

```

<beliefSet id="b1" type="Balances" extends="OpenWorld">
    <field declarationType="key" type="int" name="account"/>
    <field declarationType="value" type="int" name="balance"/>
    ...
</beliefSet>

```

Figure B.31 Balances beliefSet

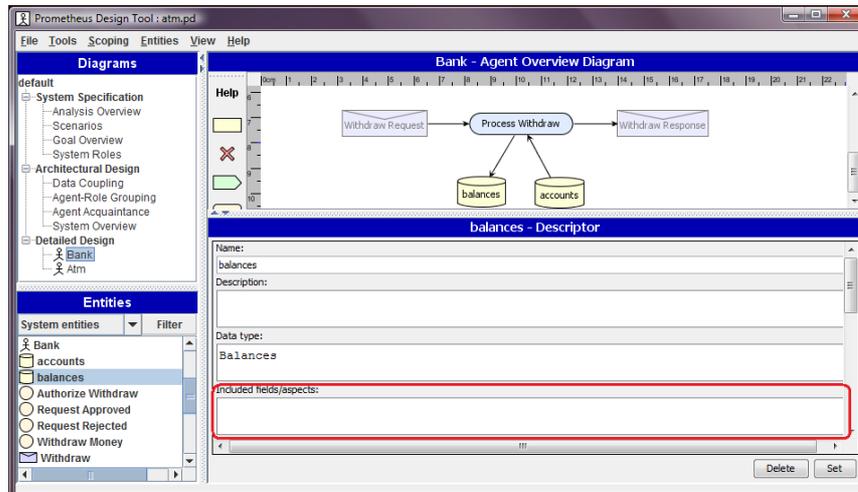


Figure B.32 Balances descriptor

We added *accounts* and *balances* to the *included fields/aspects* to the *Balances* data in Prometheus.

Table B.3 shows that the *RulePJ3cc1* rule identified that there are missing traceability relations between *ProcessWithdraw*, *WithdrawApproved*, *WithdrawCash* and *WithdrawRejected* plans in JACK and plans in Prometheus. Table B.3 also shows that the *RulePJ3cc2* rule identified that there are missing traceability relations between *Process Withdraw*, *Withdraw Approved*, *Withdraw Cash* and *Withdraw Rejected* plans in Prometheus and plans in JACK. Based on this information we can determine that are missing traceability relations between *ProcessWithdraw* in JACK and *ProcessWithdraw* in Prometheus. Plans in JACK and plans in Prometheus are related when the name of the plan in JACK and the name of plan in Prometheus are synonyms, and the name of the element that triggers the plan in Prometheus and the name of the event that the plan in JACK handles are synonyms.

We observed that *ProcessWithdraw* plan in JACK handles *WithdrawRequest* event (see Figure B.33) while no trigger properties has been defined to the *Process Withdraw* plan in Prometheus (see Figure B.34).

```
<plan id="p1" name="ProcessWithdraw" extends="Plan">
<handlesEvent type="WithdrawRequest" ref="event" />
...
</plan>
```

Figure B.33 ProcessWithdraw plan

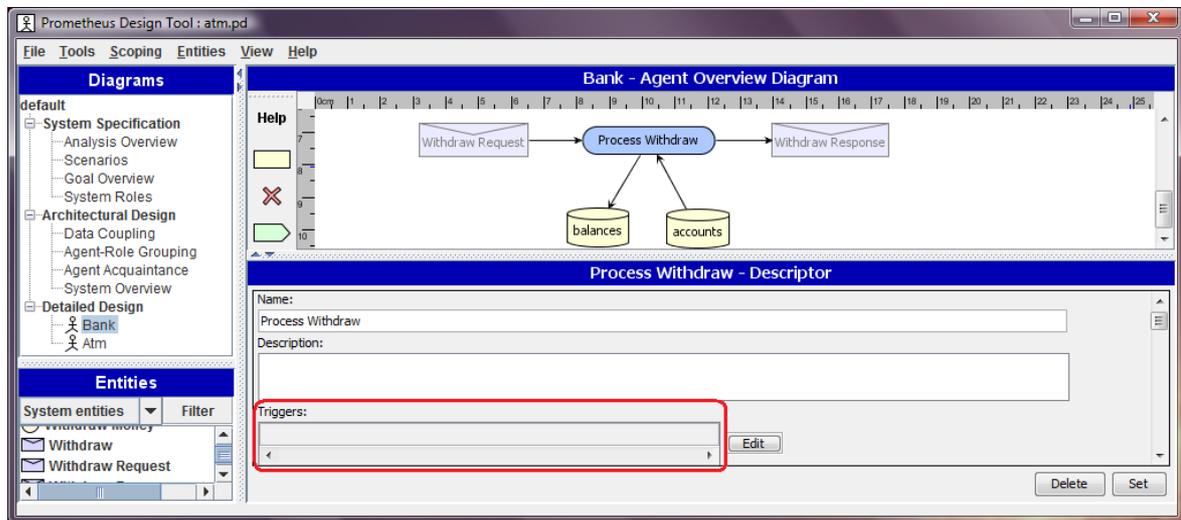


Figure B.34 Process Withdraw descriptor

We added *Withdraw Request* message to the triggers properties of the *Process Withdraw* plan. In the same way, we identified that a traceability relations between *ProcessWithdraw* plan in JACK and *ProcessWithdraw* plan in Prometheus was missing. We identified that the *WithdrawRequest* plan in JACK handles *WithdrawResponse* event (see Figure B.35) while no trigger properties has been defined to the *Withdraw Approved* plan in Prometheus (see Figure B.36).

```
<plan id="p2" name="WithdrawApproved" extends="Plan">
  <handlesEvent type="WithdrawResponse" ref="event" />
  ...
</plan>
```

Figure B.35 WithdrawApproved plan

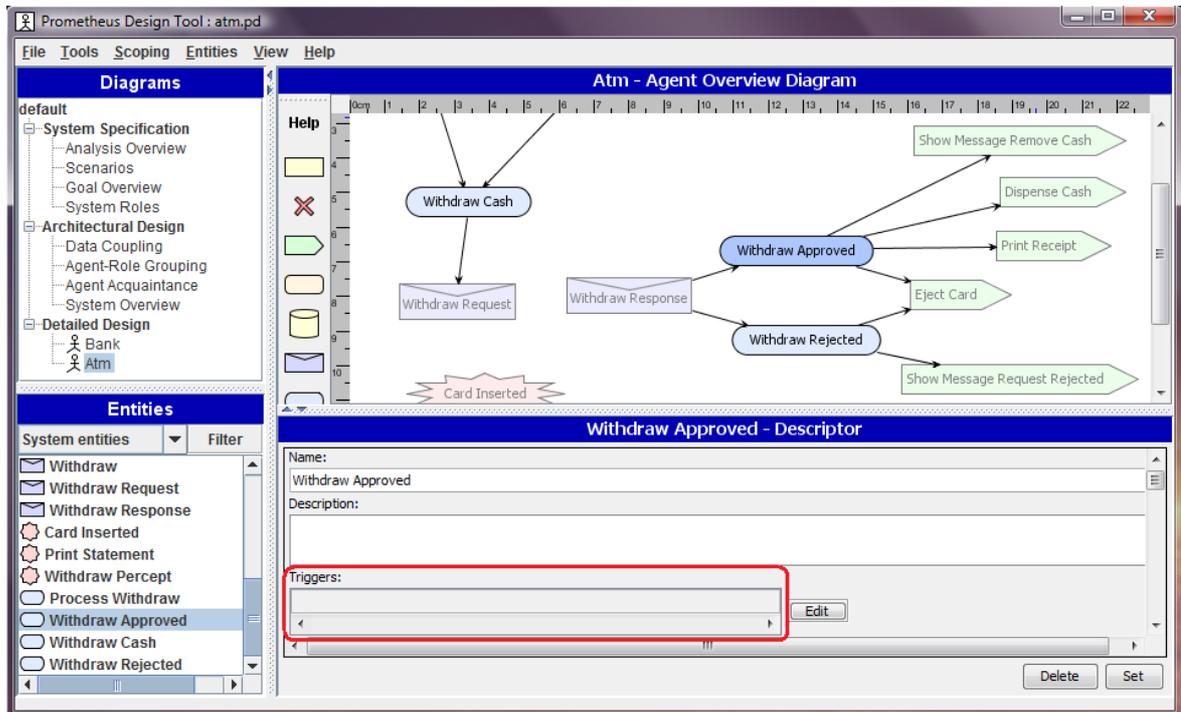


Figure B.36 Withdraw Approved descriptor

We added *Withdraw Response* message to the triggers properties of the *Withdraw Approved* plan. A traceability relation was also missing between *Withdraw Cash* plan in JACK and *Withdraw Cash* in Prometheus.

We found that *WithdrawCash* plan in JACK handles *Withdraw* event (see Figure B.37) while no trigger properties has been defined to the *Withdraw Cash* plan in Prometheus (see Figure B.38).

```

<plan id="p3" name="WithdrawCash" extends="Plan">
  <import>gui.AtmClient</import>
  <import>gui.AtmInterface</import>
  <handlesEvent type="Withdraw" ref="event" />
  ...
</plan>

```

Figure B.37 WithdrawCash plan

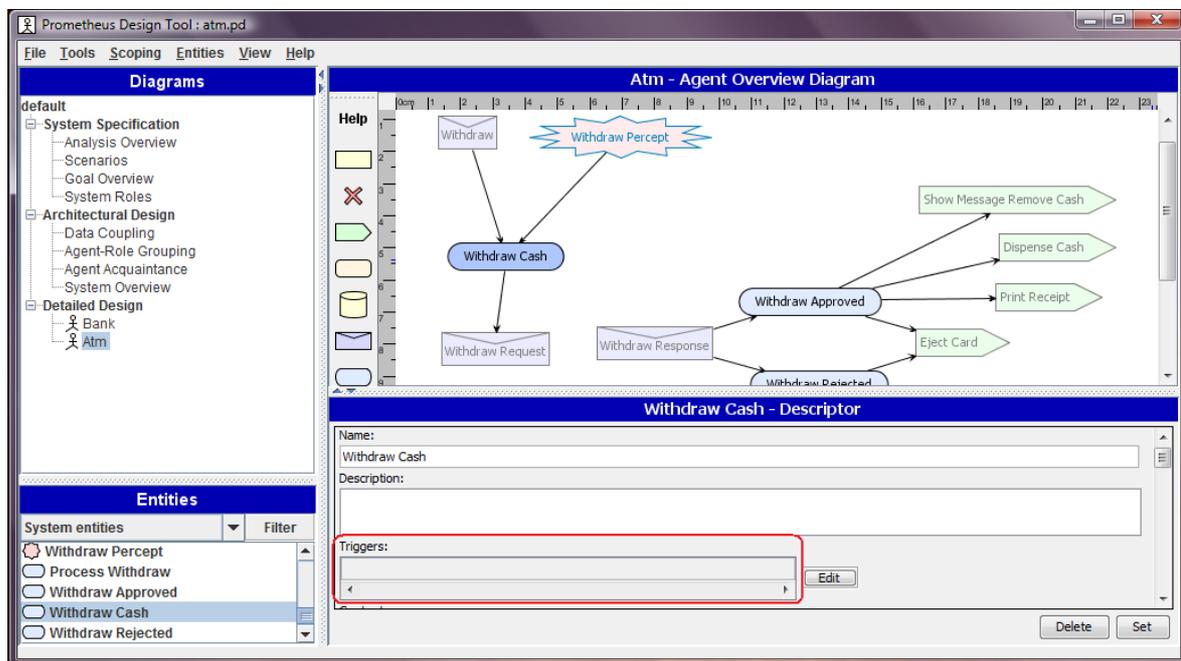


Figure B.38 Withdraw Cash descriptor

We added *Withdraw* message to the triggers properties of the *Withdraw Cash* plan. A traceability relation was missing between *Withdraw Rejected* plan in JACK and *Withdraw Rejected* plan in Prometheus. We found that *WithdrawRejected* plan in JACK handles *WithdrawResponse* event (see Figure B.39) while no trigger properties has been defined to the *Withdraw Rejected* plan in Prometheus (see Figure B.40).

```

<plan id="p4" name="WithdrawRejected" extends="Plan">
  <import>gui.AtmClient</import>
  <import>gui.AtmInterface</import>
  <handlesEvent type="WithdrawResponse" ref="event" /> ...
</plan>

```

Figure B.39 WithdrawReject plan

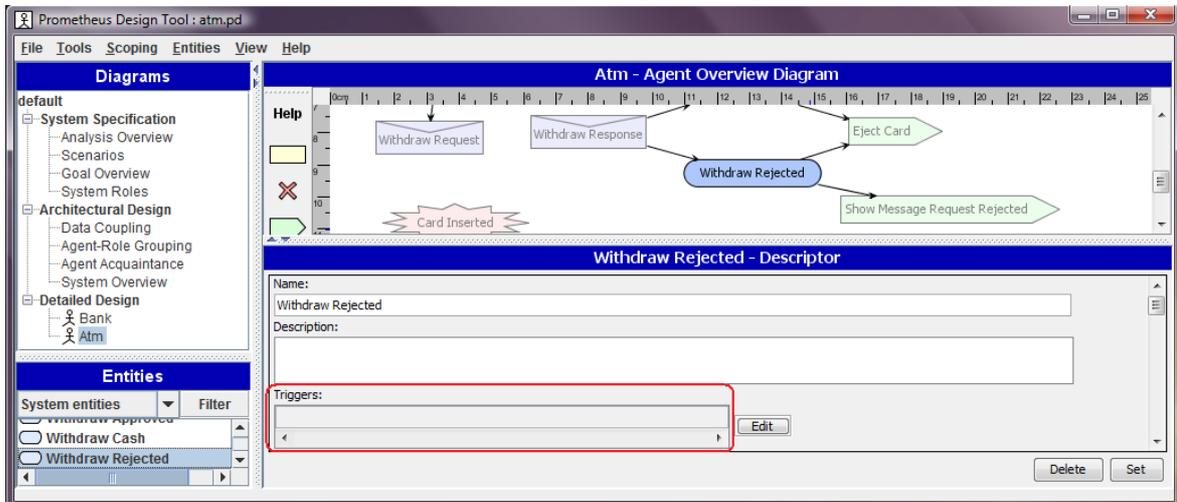


Figure B.40 Withdraw Rejected descriptor

We added *Withdraw Response* message to the triggers properties of the *Withdraw Rejected* plan.

Table B.3 shows that the *RulePJ5cc1* rule identified that is missing a traceability relation between *Request Approved*, *Request Rejected*, *Withdraw Money* and *Authorize Withdraw* goals in Prometheus and some agent in JACK. We look in the Prometheus and found that we haven't defined what Prometheus agents achieve *Request Approved*, *Request Rejected*, *Withdraw Money* and *Authorize Withdraw* goals. We updated the model and defined that *Request Approved*, *Request Rejected*, and *Withdraw Money* goals in Prometheus are achieved by *Atm* agent and *Authorize Withdraw* is achieved by *Bank* agent.

We run the prototype tool again and compared the results with relations identified by the tool calculate precision and recall again. We found the 33 traceability relations that were missing and the precision and recall calculated were 100% and 100%, respectively.

Appendix C – Air Traffic Control Environment

C.1 Introduction

This section describes the development of a multi-agent system to implement the Air Traffic Control Environment used as a case study to evaluate our approach to generate traceability relations automatically and to identify missing elements between artefacts created during the development of a multi-agent system.

Air traffic congestion is a global issue and several air traffic management systems have already been built to alleviate this problem [Liunberg 1992]. The air control environment consists of a system that implements arrival sequencing for an airport. The main goal of an air control environment is to find the best landing time for an aircraft in order to alleviate congestion and its associated delays.

A feeder airport has the responsibility to process traffic of aircrafts. A Feeder airport contains information about all aircraft schedule arrivals that consists of the call sign (unique identifier of an aircraft used in the radio communications), booking time, ETA (Estimate Times of Arrival) to use for booking, the arrival time at destination control area, and the ETA at control area entrance. The feeder airport waits until the booking time has passed and then sends the information to destination airport. A feeder aircraft receives update information about schedule changes such as a takeoff discard of an aircraft.

An aircraft sends a message to the airport when enter control area of the airport destination and waits until a runway has been allocated. To find the best landing time for an aircraft, the airport manager first queries all runway managers for the “best landing time” for an aircraft and then chooses one. After the airport manager notifies the decision to the runway manager and to the aircraft. In order to maximizing landing, faster aircraft that arrive later to the airport control area, push out earlier already assigned slower aircraft. A new bidding occurs to allocate a runway slot for the slower aircraft. During the approaching to landing, the aircraft test continually to see if the runaway still allocated for landing until the landing time (ATL) has passed.

C.2 Organizational Models

The Air Traffic Environment is composed of Aircraft, Airport, Feeder and Runway actors. Figure C.1 shows the actors and its strategic dependencies relationships. The Aircraft actor depends on the Airport actor to have ATL resource provided. The Feeder actor depends on the Airport actor to have Find Best Landing Time for an Aircraft goal achieved. The Airport actor depends on Runway actor to have Allocate Runway Slot goal achieved. The Airport actor depends on Runway actor to have Slot Allocated resource provided.

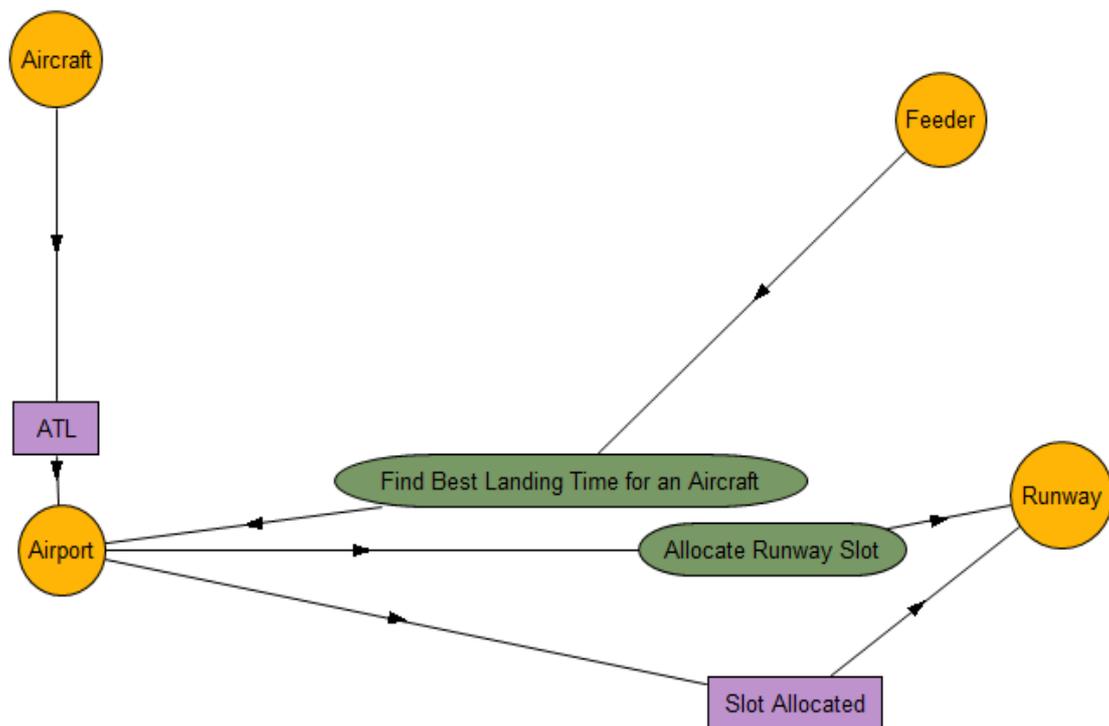


Figure C.1 Strategic Dependency model for Air Traffic Environment

The Landing task of the Aircraft actor is decomposed on Assign Slot, Initiate Approach and Follow Approach tasks. The Follow Approach task uses Landing Information resource and the Assign Slot task depends on the Airport actor to provide the ATL resource in order to perform Assign Slot task.

The Process Schedule for a Feeder task of the Feeder actor is decomposed on TakeOff and TakeOffDiscard tasks. The TakeOff task is decomposed on Request Booking task. The Request Booking task depends on the Airport actor to achieve Find Best Landing Time for an Aircraft goal.

The Airport actor performs Request Runway task as means to achieve Find Best Landing Time for an Aircraft goal. The Request Runway task uses ATL resource and it depends on Runway actor to achieve Allocate Runway Slot goal and to provide the Slot Allocated resource.

The Runway actor performs Respond Runway Request task as means to achieve Allocate Runway Slot goal. The Respond Runway Request task uses Slot Allocated resource.

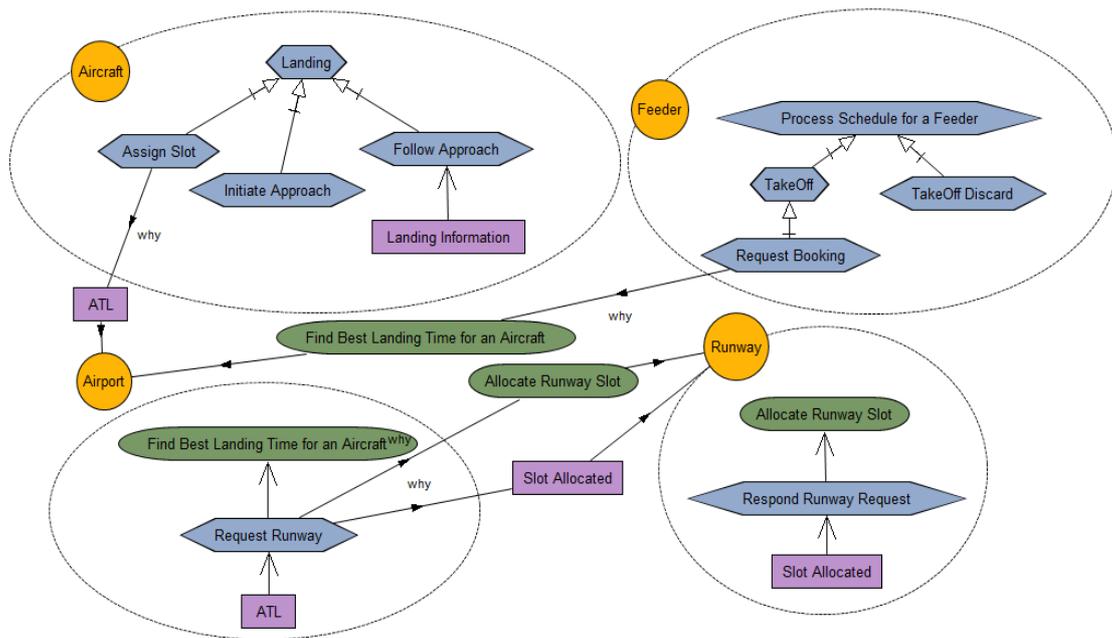


Figure C.2 Strategic Rationale model for Air Traffic Environment

C.3 Prometheus Models

Figure C.3 shows the Goal diagram for the Air Traffic Control Environment. In the top level, we have Landing, Find Best Land Time for an Aircraft, Schedule Arrival for a Feeder, Process Schedule for a Feeder, Assign Runway, Request Slot, and Assign Slot. The Find Best Land time for an Aircraft goal is refined by Query Best Landing Time from All Runway Manager and Push Out goals. The Landing goal is refined by Initiate Aircraft Approach and Progresses an aircraft to Landing goals.

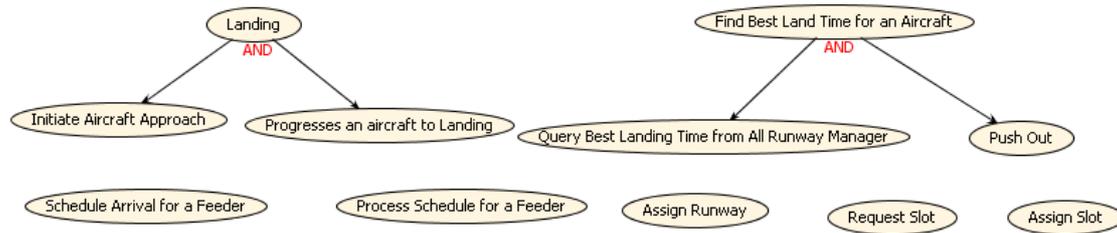


Figure C.3 Goal diagram for Air Traffic Environment

The Air Traffic Control Environment application consists of a multi-agent systems composed of four agents: Aircraft, Airport, Feeder, and Runway.

The Feeder agent has the Traffic Feeding capability. The Traffic Feeding capability overview diagram is shown in the Figure C.4. The Traffic Feeding capability handles Aircraft Event and Traffic Event events, sends Aircraft Event event, contains the Schedule data, and uses Traffic, Takeoff, and Takeoff Discard plan.

Traffic and Takeoff plans handle Traffic Event message. Initially, a Traffic Event message is posted when the Feeder agent is created. The Traffic Event message contains information (schedule rows) about all aircraft schedule arrivals that consists of the call sign (unique identifier of an aircraft used in the radio communications), booking time, ETA (Estimate Times of Arrival) to use for booking, the arrival time at destination control area, and the ETA at control area entrance. To each schedule row in the Traffic Event message, the Traffic plan wait until the booking time has passed and then post a Traffic Event message passing the schedule row, airport and destination. This time the Traffic Event message is handled by the Takeoff plan. The Takeoff plan sends a booking request (Aircraft Event message) to the Airport. The Takeoff plan waits for the arrival time of the aircraft to then creates an Aircraft agent. The Takeoff Discard plan handles Aircraft Event message sent back from the Airport agent for the booking.

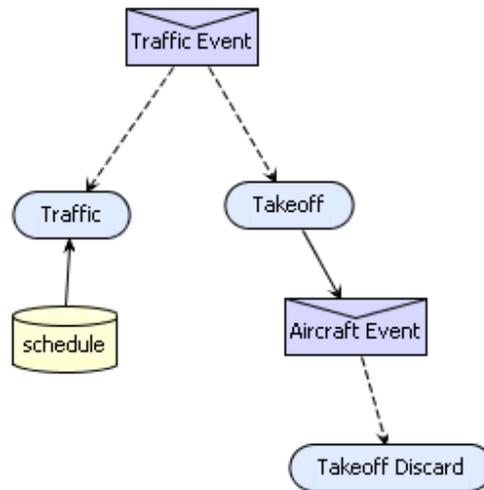


Figure C.4 Traffic Feeding Capability

The Airport agent has the Arrival Sequencing capability (see Figure C.5). The Arrival Sequencing capability contains the Request Slot Plan plan and the Semaphore data. The Request Slot Plan plan handles the Aircraft Event message and propagates it to all available runways in order to find the best one and then notifies the decision to the runway and to the aircraft.



Figure C.5 Arrival Sequencing Capability

The Runway agent has the Runway Assigning capability (see Figure C.6). The Runway Assigning capability contains Runway Request and Runway Assign plans, RunwayInfo data and handles the Aircraft Event message. The Runway Request plan handles the Aircraft Event message sent from the Airport agent. The Runway Request plan checks all runway assignments to find the first slot that is not used or used to a slower aircraft (slower aircraft are push out). The Aircraft agent sends an Aircraft Event message assigning the runway chosen. The Runway Assign plan handles this Aircraft Event message and allocates the slot to the runway. If the slot is already occupied then it is re-allocated and then an Aircraft Event message is sent to the Airport agent requesting a new booking to the Aircraft that has been push out. An Aircraft Event message is also sent to the Aircraft agent confirming the allocation of the runway.

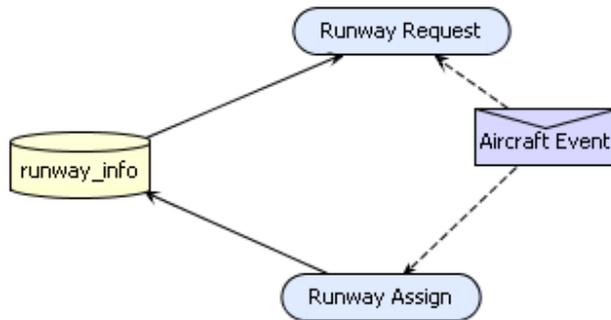


Figure C.6 Runway Assigning Capability

The Aircraft agent has the Flying capability (see Figure C.7). The Flying capability uses Monitor Aircraft, Initial Approach, Follow Approach, and Assign Slot Plan plans, has LandingInfo data, and handles Aircraft Event, Enter Control Area, Approaching messages; post Enter Control Area, Approaching events. The Monitor Aircraft plan handles Enter Control Area message. The Monitor Aircraft plan sends an Aircraft Event message confirming landing at arrival ETA and post Approaching message. Initially, the Initial Approach plan handles the Approaching message. The Initial Approach plan wait until the runway has been allocated and then post an Approaching message. This time, the Follow Approach message handles the Approaching message and test continually to see if the runway still allocated for landing until the landing time (ATL) has passed. The Assign Slot plan handles the notification of Aircraft Event message sent by the Airport passing the runway information that the Aircraft has been allocated. The Assign Slot plan adds a new fact to the LandingInfo beliefset with the runway and ATL data.

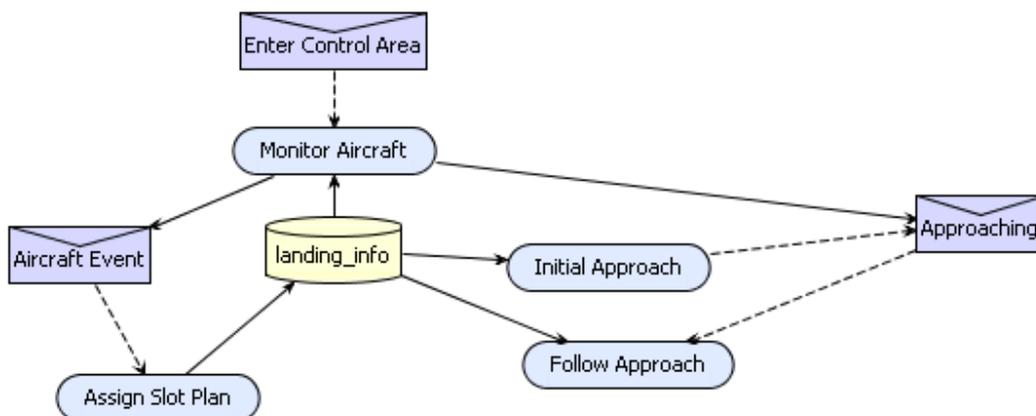


Figure C.7 Flying Capability

Traffic and Takeoff plans handle Traffic Event message. Initially, a Traffic Event message is posted when the Feeder agent is created. The Traffic Event message contains information (schedule rows) about all aircraft schedule arrivals that consists of the call sign (unique identifier of an aircraft used in the radio communications), booking time, ETA (Estimate Times of Arrival) to use for booking, the arrival time at destination control area, and the ETA at control area entrance. To each schedule row in the Traffic Event message, the Traffic plan wait until the booking time has passed and then post a Traffic Event message passing the schedule row, airport and destination. This time the Traffic Event message is handled by the Takeoff plan. The Takeoff plan sends a booking request (Aircraft Event message) to the Airport and wait for the arrival time of the aircraft then creates an Aircraft agent. The Takeoff Discard plan handles Aircraft Event message sent back from the Airport agent for the booking.

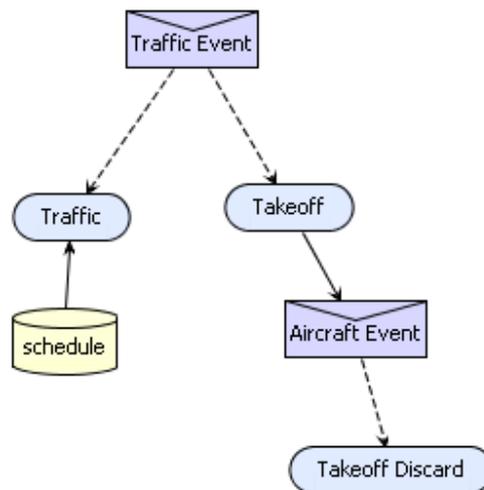


Figure C.8 Traffic Feeding Capability

The Airport agent has the Arrival Sequencing capability (see Figure C.9). The Arrival Sequencing capability contains the Request Slot Plan and the Semaphore data. The Request Slot Plan plan handles the Aircraft Event message and propagates it to all available runways in order to find the best one and then notifies the decision to the runway and the aircraft.



Figure C.9 Arrival Sequencing Capability

The Runway agent has the Runway Assigning capability (see Figure C.10). The Runway Assigning capability contains Runway Request and Runway Assign plans, RunwayInfo data and handles the Aircraft Event message. The Runway Request plan handles the Aircraft Event message sent from the Airport agent. The Runway Request plan checks all runway assignments to find the first slot that is not used or used to a slower aircraft (slower aircraft are push out). The Aircraft agent sends an Aircraft Event message assigning the runway chosen. The Runway Assign plan handles this Aircraft Event message and allocates the slot to the runway. If the slot is already occupied then it is re-allocated and then an Aircraft Event message is sent to the Airport agent requesting a new booking to the Aircraft that has been push out. An Aircraft Event message is also sent to the Aircraft agent confirming the allocation of the runway.

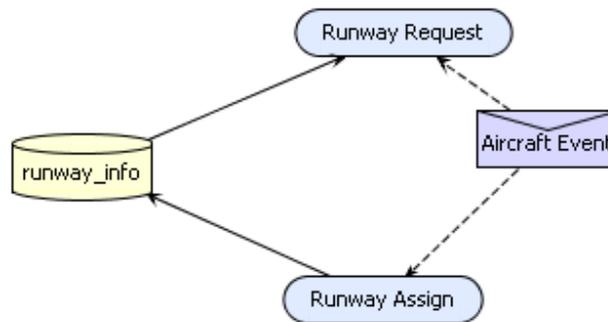


Figure C.10 Runway Assigning Capability

The Aircraft agent has the Flying capability (see Figure C.11). The Flying capability uses Monitor Aircraft, Initial Approach, Follow Approach, and Assign Slot Plan plans, has LandingInfo data, and handles Aircraft Event, Enter Control Area, Approaching messages; post Enter Control Area, Approaching events. The Monitor Aircraft plan handles Enter Control Area message. The Monitor Aircraft plan sends an Aircraft Event message confirming landing at arrival ETA and post Approaching message. Initially, the Initial Approach plan handles the Approaching message. The Initial Approach plan wait until the runway has been allocated and then post an Approaching message. This time, the Follow Approach message handles the Approaching message and test continually to see if the runaway still allocated for landing until the landing time (ATL) has passed. The Assign Slot plan handles the notification of Aircraft Event message sent by the Airport passing the runway information that the Aircraft has been allocated. The Assign Slot plan adds a new fact to the LandingInfo beliefset with the runway and ATL data.

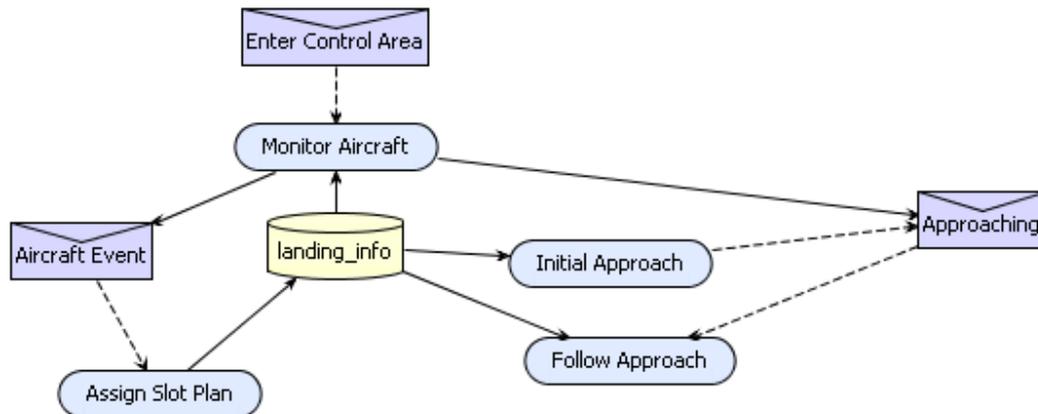


Figure C.11 Flying Capability

C.4 JACK Code

The multi-agent system was implemented using JACK Agent Language. The system consists of Aircraft, Airport, Feeder and Runway agents, LandingInfo and RunwayInfo beliefSets, ArrivalSequencing, Flying, RunwayAssigning and TrafficFeeding capabilities, AircraftEvent, Approaching, EnterControlArea, and TrafficEvent events, AssignSlot, FollowApproach, InitialApproach, MonitorAircraft, RequestSlot, RunwayAssign, RunwayRequest, Takeoff, TakeoffDiscard, and Traffic plans.

Figure C.12 shows Aircraft agent. The Aircraft agent has Flying capability.

```

/** Aircraft agents. */
agent Aircraft extends Agent {
  #has capability Flying fly;

  Aircraft(String id,String airport,long eta){
    super(id);
    fly.start(id,airport,eta);
  }
}

```

Figure C.12 Aircraft agent

Figure C.13 shows Airport agent. The Airport agent has ArrivalSequencing capability.

```

import java.util.Hashtable;
import aos.jack.jak.event.TracedMessageEvent;

/** Airport agents.*/
agent Airport extends Agent {

    #has capability ArrivalSequencing seq;

    Airport(String name,String [] runway){
        super(name);

        for (int i = 0; i<runway.length; i++)
            new Runway(runway[i],i);
        seq.enable(runway);
        TracedMessageEvent.tracer.start(this);
    }
}

```

Figure C.13 Airport agent

Figure C.14 shows *Feeder* agent. The *Feeder* agent has *ArrivalSequencing* capability.

```

/** The Feeder agents model source airports and other "sources of
    aircraft". Each feeder agent has it's own schedule. */
agent Feeder extends Agent {
    #has capability TrafficFeeding feed;

    Feeder(String name,String destination){
        super(name);
        feed.load(name,destination);
    }
}

```

Figure C.14 Feeder agent

Figure C.15 shows *Runway* agent. The *Runway* agent has *RunwayAssigning* capability.

```

import aos.jack.jak.event.TracedMessageEvent;

/**Runway agents.*/
agent Runway extends Agent {
    #has capability RunwayAssigning assign;

    Runway(String name,int index){
        super(name);
        assign.setName(name,index);
        TracedMessageEvent.tracer.start(this);
    }
}

```

Figure C.15 Runway agent

Figure C.16 shows *LandingInfo* beliefSet that contains *runway* and *ATL* fields and the query function *get*.

```
/** Relation LandingInfo is used to keep the landing
information. */
beliefset LandingInfo extends ClosedWorld {
    #value field String runway;
    #value field long ATL;

    #linear query get(logical String runway,logical long ATL);
}
```

Figure C.16 LandingInfo beliefSet

Figure C.17 shows *RunwayInfo* beliefSet that contains *ATL*, *aircraft*, *ETA*, and *booking* fields. The *ATL* field is declared as key field. The *RunwayInfo* beliefSet has *gui* member field of Stack. The *RunwayInfo* beliefSet contains query function *usingSlot* and *slotUsed*. The *RunwayInfo* beliefSet implements *newfact*, and *delfact* callback methods.

```
import aos.jack.jak.util.timer.DilationController;

/** Relation RunwayInfo is used for keeping the current usage of a
runway. */

beliefset RunwayInfo extends ClosedWorld {
    #key field long ATL;
    #value field String aircraft;
    #value field long ETA;
    #value field boolean booking;

    Stack gui;

    void setName(String name,int index) {
        gui = new Stack(name,index);
    }

    #indexed query
    usingSlot(logical long ATL,
              String aircraft,
              logical long ETA,
              logical boolean booking);

    #indexed query
    slotUsed(long ATL,
             logical String aircraft,
             logical long ETA,
             logical boolean booking);

    public void newfact(Tuple t,BeliefState is,BeliefState was)
    {
        if (gui == null)
            return;
    }
}
```

```

    RunwayInfo__Tuple info = (RunwayInfo__Tuple)t;
    gui.addRow(info.ATL, info.aircraft+
        " ["+DilationController.timeString(info.ETA)+" ]");
}

public void delfact(Tuple t, BeliefState was) {
    if (gui == null)
        return;
    RunwayInfo__Tuple info = (RunwayInfo__Tuple)t;
    gui.removeRow(info.ATL);
}
}

```

Figure C.17 RunwayInfo beliefSet

Figure C.18 shows *ArrivalSequencing* capability that handles *AircraftEvent* event, has *mutex* data and uses *RequestSlot* plan. The *ArrivalSequencing* capability has *runways* member array of Strings. The *ArrivalSequencing* capability implements *getRunways()* and *enable()* methods.

```

import aos.jack.util.thread.Semaphore;

/** The ArrivalSequencing capability contains the handling of
landing requests from aircraft through negotiation with available
runways for an appropriate landing allocation.
*/
public capability ArrivalSequencing extends Capability {

    #handles external event AircraftEvent;
    #private data Semaphore mutex();
    #uses plan RequestSlot;

    String [] runways;

    String [] getRunways(){
        return runways;
    }

    void enable(String [] runways){
        this.runways = runways;
        mutex.signal();
    }
}

```

Figure C.18 ArrivalSequencing capability

Figure C.19 shows *Flying* capability that handles *AircraftEvent*, *EnterControlArea*, and *Approaching* events, has *LandingInfo* data. The *Flying* capability sends *AircraftEvent* event, and post *EnterControlArea* and *Approaching* events. The *Flying* capability uses *MoniotrAircraft*, *FollowApproach*, *InitialApproach*, and *AssignSlot* plans.

```

/** The Flying capability contains the tracking of the
approach from when the aircraft enters the destination
airport control area. */

public capability Flying extends Capability {

    #private data LandingInfo landing_info();
    #handles external event AircraftEvent;
    #sends event AircraftEvent;
    #handles event EnterControlArea;
    #handles event Approaching;

    #posts event EnterControlArea enter;
    #posts event Approaching follow;

    void start(String id,String airport,long eta){
        postEvent(enter.start(id,airport,eta));
    }

    #uses plan MonitorAircraft;
    #uses plan FollowApproach;
    #uses plan InitialApproach;
    #uses plan AssignSlot;
}

```

Figure C.19 Flying Capability

Figure C.20 shows *RunwayAssigning* capability that handles *AircraftEvent* event, has *RunwayInfo* data and uses *RunwayRequest* and *RunwayAssign* plans. The *RunwayAssigning* capability has *SLOTGAP* constant and implements *slotTime* and

```

/** The RunwayAssigning capability contains the bidding side of the
runway assignment negotiation. */
public capability RunwayAssigning extends Capability {

    #handles external event AircraftEvent;
    #private data RunwayInfo runway_info();
    final static long SLOTGAP = 180000; // 3 minutes = 180000
    milliseconds

    static long slotTime(long time    {
        long x = time/SLOTGAP;
        return (x+1)*SLOTGAP;
    }

    void setName(String name,int index){
        runway_info.setName(name,index);
    }

    #uses plan RunwayRequest;
    #uses plan RunwayAssign;
}

```

Figure C.20 Runway Assigning Capability

setName methods.

Figure C.21 shows *TrafficFeeding* capability that handles *AircraftEvent* and *TrafficEvent* event, posts *TrafficEvent*, sends *AircraftEvent*, has *Schedule* data and uses *Traffic*, *Takeoff*, and *TakeoffDiscard* plans. *RunwayRequest* and *RunwayAssign* plans. The *RunwayAssigning* capability has *gui* member field of *TrafficGUI* and implements *load* method.

```
/** The TrafficFeeding capability contains the processing of a
    departure schedule.
 */
capability TrafficFeeding extends Capability {

    #handles external event AircraftEvent;
    #sends event AircraftEvent request;
    #handles event TrafficEvent;
    #private data Schedule schedule();
    #uses plan Traffic;
    #uses plan Takeoff;
    #uses plan TakeoffDiscard;

    TrafficGUI gui;

    #posts event TrafficEvent traffic;

    void load(String name,String destination){
        System.err.println("Feed from "+name+" opened.");
        schedule.load(name+".dat",new TrafficGUI(name));
        postEvent(traffic.open(name,destination));
    }
}
```

Figure C.21 TrafficFeeding capability

Figure C.22 shows Aircraft event has run aircraft, ETA, ATL, booking, and mode data members and implements assign and confirm posting methods. The AircraftEvent event has REQUEST, ASSIGN, and NOTIFIES constants and implements name and toString methods.

```
import aos.jack.jak.util.timer.DilationController;

/**
    The event AircraftEvent is used in the messaging between aircraft
    and airport.
 */
event AircraftEvent extends TracedMessageEvent {
    String runway;
    String aircraft;
    long ETA;
    long ATL;
```

```

#posted as
    request(String aircraft,long ATL,long ETA,boolean booking){
        this.runway = null;
        this.aircraft = aircraft;
        this.ATL = ATL;
        this.ETA = ETA;
        this.booking = booking;
        mode = REQUEST;
        message =
            (booking? "Booking " : "Request ") +
            name(aircraft) +
            " ETA= "+DilationController.timeString(ETA);
    }

#posted as
    assign(String runway,long ATL,String aircraft,long ETA,boolean
booking) {
        this.runway = runway;
        this.aircraft = aircraft;
        this.ATL = ATL;
        this.ETA = ETA;
        mode = ASSIGN;
        this.booking = booking;
        message =
            name(aircraft) +
            (booking? " booked " : " assigned ")
            +name(runway)+" "+
            DilationController.timeString(ATL);
    }

#posted as
    confirm(String runway,long ATL,String aircraft){
        this.runway = runway;
        this.aircraft = aircraft;
        this.ATL = ATL;
        mode = NOTIFY;
        message =
            "Scheduled "+
            name(aircraft)+" "+
            DilationController.timeString(ATL);
    }

    static String name(String a){
        int i = a.indexOf('@');
        return (i == -1)? a : a.substring(0,i);
    }

    public String toString(){
        return message;
    }
}

```

Figure C.22 AircraftEvent event

Figure C.23 shows *Approaching* event. The *Approaching* event has *runway*, *aircraft*, *ETA*, *ATL*, *booking*, and *mode* data members and implements *assign* and *confirm* posting methods. The *AircraftEvent* event has *REQUEST*, *ASSIGN*, and *NOTIFIES* constants and implements *name* and *toString* methods.

```

/** The Approaching event marks for an aircraft the period from
entering the control area of the destination airport to the landing
of the aircraft. */

event Approaching extends BDIGoalEvent {

    logical String runway;
    logical long ATL;

    #posted as
    approach(logical String runway,logical long ATL)
    {
        this.runway = runway;
        this.ATL = ATL;
    }

    #set behavior ApplicableExclusion none ;
    /* defines how failed plans are excluded from being applicable. The
value is "none" or one or both of "failed" and "rank". "none" means
that plan failure is forgotten immediately, and a failed plan will
turn up as applicable (if it is). */

}

```

Figure C.23 Approaching event

Figure C.24 shows *EnterControlArea* event. The *EnterControlArea* event has *id*, *airport*, and *eta* data members and implements *start* posting methods.

```

/** The EnterControlArea event marks for an aircraft that it enters
the control area of the destination airport. */
event EnterControlArea extends Event {
    String id;
    String airport;
    long eta;

    #posted as
    start(String id,String airport,long eta){
        this.id = id;
        this.airport = airport;
        this.eta = eta;
    }
}

```

Figure C.24 EnterControlArea

Figure C.25 shows *TrafficEvent* event. The *TrafficEvent* event has *row*, *airport*, *destination*, *mode*, data members and *SCHEDULE* and *AIRCRAFT* constants. The *TrafficEvent* event implements *open* and *run* methods.

```

/** A TrafficEvent event marks events in the feeder traffic. An
initial TrafficEvent.SCHEDULE is posted at agent construction, and
then TrafficEvent.AIRCRAFT events are posted for the schedule rows.*/
event TrafficEvent extends Event {

    ScheduleRow row;
    String airport;
    String destination;
    int mode;
    final static int SCHEDULE = 0;
    final static int AIRCRAFT = 1;

    #posted as
    open(String airport,String destination){
        this.airport = airport;
        this.destination = destination;
        mode = SCHEDULE;
    }

    #posted as
    run(ScheduleRow row,String airport,String destination){
        this.row = row;
        this.airport = airport;
        this.destination = destination;
        mode = AIRCRAFT;
    }
}

```

Figure C.25 TrafficEvent event

Figure C.26 shows AssignSlot plan. The *AssignSlot* plan handles *AircraftEvent* event and modifies *LandingInfo* data. The *AssignSlot* plan implements *body* reasoning method.

```

/** The plan AssignSlot digests an ATL notification from the airport
by updating the landing_info belief.*/

plan AssignSlot extends Plan {

    #handles event AircraftEvent ev;
    #modifies data LandingInfo landing_info;

    static boolean relevant(AircraftEvent ev){
        return ev.mode == AircraftEvent.NOTIFY;
    }
    body(){
        landing_info.add(ev.runway, ev.ATL);
    }
}

```

Figure C.26 AssignSlot plan

Figure C.27 shows *FollowApproach* plan. The *FollowApproach* plan handles *Approaching* event and reads *LandingInfo* data. The *FollowApproach* plan implements *body* and *maintain* reasoning methods.

```
import aos.jack.jak.util.timer.DilationController;

/** The FollowApproach plan progresses an aircraft to landing when a
landing allocation is provided.*/
plan FollowApproach extends Plan {

    #handles event Approaching ev;
    #reads data LandingInfo landing_info;

    context(){
        landing_info.get(ev.runway, ev.ATL);
    }

    body(){
        System.err.println(agent.name()+" following approach to runway "+
ev.runway+" at "+DilationController.timeString(ev.ATL.getValue()));
        @maintain(landing_info.get(ev.runway, ev.ATL),
            @waitFor(afterMillis(ev.ATL.getValue())));
    }
}
```

Figure C.27 FollowApproach plan

Figure C.28 shows *InitialApproach* plan. The *InitialApproach* plan handles *Approaching* event, posts *Approaching* event and reads *LandingInfo* data. The *InitialApproach* plan implements *context* and *body* reasoning methods.

```
/** The InitialApproach plan progresses an aircraft to landing during
the initial phase when the landing allocation is not yet provided. */
plan InitialApproach extends Plan {

    #handles event Approaching ev;
    #posts event Approaching land;
    #reads data LandingInfo landing_info;

    context(){
        landing_info.nFacts() == 0;
    }

    body(){
        logical String r;
        logical long a;
        @waitFor(landing_info.get(r, a));
        @achieve(landing_info.get(ev.runway, ev.ATL) &&
            afterMillis(ev.ATL.getValue()),
            land.approach(ev.runway, ev.ATL));
    }
}
```

Figure C.28 InitialApproach plan

Figure C.29 shows MonitorAircraft plan. The MonitorAircraft plan handles EnterControlArea event, sends AircraftEvent event, posts Approaching event and reads LandingInfo data. The MonitorAircraft plan has runway and ATL field members and implements body reasoning method.

```
import aos.jack.jak.util.timer.DilationController;

/** The MonitorAircraft plan is invoked for monitoring the flight of
an arriving aircraft, from entering the control area to its landing.*/

plan MonitorAircraft extends Plan {

    #handles event EnterControlArea ev;
    #sends event AircraftEvent req;
    #posts event Approaching land;
    #reads data LandingInfo landing_info;
    logical String runway;
    logical long ATL;

    body(){
        // Confirm landing at arrival ETA
        @send(ev.airport, req.request(ev.id, ev.eta, ev.eta, false));
        @achieve(landing_info.get(runway, ATL) &&
            afterMillis(ATL.getValue()), land.approach(runway, ATL));

        System.err.println(ev.id+" landed on runway "+runway.getValue()
            +" for "+DilationController.timeString(ATL.getValue()+ " at "+
            DilationController.timeString(getAgent().timer.getTime()));
    }
}
```

Figure C.29 MonitorAircraft plan

Figure C.30 shows RequestSlot plan. The RequestSlot plan handles AircraftEvent event, sends AircraftEvent event, posts Approaching event and reads LandingInfo data. The *MonitorAircraft* plan has *runway* and *ATL* field members and implements *body* reasoning method.

```
import aos.jack.util.thread.Semaphore;

/** Plan RequestSlot handles an AircraftEvent.REQUEST by propagating it
to all available runways, collecting all their suggestions, choosing
the best one, and then notifying the runway and aircraft concerned.*/

plan RequestSlot extends Plan {

    #handles event AircraftEvent ev;
    static boolean relevant(AircraftEvent ev){
        return ev.mode == AircraftEvent.REQUEST;
    }
    #uses interface ArrivalSequencing env;
    #modifies data Semaphore mutex;
```

```

long pushed_ETA = 0;
long ATL = -1;
String runway;

body(){
    System.err.println("Receved"+ev);
    if (env.runways == null)
        throw new Error("There are no runways??");
    AircraftEvent query =
        ev.request(ev.aircraft, ev.ATL, ev.ETA, ev.booking);
    @waitFor(mutex.planWait());
    System.err.println("Issuing"+ev);
    sendRequests(query);
    receiveReplies(query);
    notifyResult();
    mutex.signal();
}

#reasoning method
sendRequests(AircraftEvent query){
    for (int i=0; i<env.runways.length; i++)
        @send(env.runways[i], query);
}

#reasoning method
receiveReplies(AircraftEvent query){
    for (int i=0; i<env.runways.length; i++) {
        @waitFor(query.replied());
        AircraftEvent r = (AircraftEvent)query.getReply();
        if (betterSlot(r)) {
            runway = r.from;
            ATL = r.ATL;
            pushed_ETA = r.ETA;
        }
    }
}

boolean betterSlot(AircraftEvent ev){
    if ((ATL == -1) || (ev.ATL < ATL))
        return true;
    if (ev.ATL > ATL)
        return false;
    if (pushed_ETA == 0)
        return false;
    if (ev.ETA == 0)
        return true;
    return (ev.ETA > pushed_ETA);
}

#reasoning method
notifyResult(){
    @send(runway, ev.assign(runway, ATL, ev.aircraft, ev.ETA, ev.booking));
    @send(ev.from, ev.confirm(runway, ATL, ev.aircraft));
}
}

```

Figure C.30 RequestSlot plan

Figure C.31 shows *RunwayAssign* plan. The *RunwayAssign* plan modifies *RunwayInfo* data and implements *body* reasoning method. Figure C.31 shows *RunwayRequest* plan.

```

/**The RunwayAssign plan responds to an AircraftEvent.ASSIGN by
    filling an allocation for this runway. If the allocation slot is
    already occupied, the occupant is re-scheduled. Also, the aircraft
    getting an allocation is notified.*/

plan RunwayAssign extends Plan {

    #handles event AircraftEvent ev;

    static boolean relevant(AircraftEvent ev)
    {
        return ev.mode == AircraftEvent.ASSIGN;
    }

    #modifies data RunwayInfo runway_info;

    body(){
        logical String ac;
        logical long eta;
        logical boolean booking;
        if (runway_info.slotUsed(ev.ATL,ac,eta,booking)) {
            System.err.println("PUSH "+ac.getValue()+" by
"+ev.aircraft);
            @send(ev.from,
                ev.request(ac.getValue(),ev.ATL, eta.getValue(),
                    booking.getValue()));
        }
        runway_info.add(ev.ATL,ev.aircraft,ev.ETA,ev.booking);
        @send(ev.from,ev.confirm(agent.name(),ev.ATL,ev.aircraft));
    }
}

```

Figure C.31 RunwayAssign plan

Figure C.32 shows *RunwayRequest* plan handles *AircraftEvent* event reads *RunwayInfo* data, uses *RunwayAssigning* interface and reads *RunwayInfo* data and implements *relevant*, *body*, *freeSlot*, *runway_info.remove*, *cleanUp*, and *timestring* reasoning methods.

```

import aos.jack.jak.util.timer.DilationController;
/** The RunwayRequest plan responds to an AircraftEvent.REQUEST by
    suggesting an allocation for this runway. The allocation inspects
    all time slots from the given ETA, to find the first that is
    unused, or used with an allocation of "lesser importance".
    + If this request is an early booking, then it may push a previous
    earlier booking if this ETA is prior to the earlier booking's
    ETA.
    + If this request is the arrival request, then it may push an
    earlier booking (regardless), or a previous arrival assignment if
    this ETA is prior to the previous assignment's ETA.*/

```

```

plan RunwayRequest extends Plan {

    #handles event AircraftEvent ev;
    #reads data RunwayInfo runway_info;
    #uses interface RunwayAssigning env;

    static boolean relevant(AircraftEvent ev){
        return ev.mode == AircraftEvent.REQUEST;
    }

    long pushed_eta = 0; // Local data assigned by freeSlot()

    body(){
        String me = agent.name();
        long t;
        cleanUp(ev.aircraft);
        for (t = env.slotTime(ev.ETA); true; t += env.SLOTGAP) {
            if (freeSlot(t))
                break;
        }
        @reply(ev, ev.assign(me, t, ev.aircraft, pushed_eta, ev.booking));
    }

    #reasoning method
    freeSlot(long t){
        logical long eta;
        logical String ac;
        logical boolean booking;
        pushed_eta = 0;
        if (runway_info.slotUsed(t, ac, eta, booking)) {
            System.err.println(" Check against "+ac.getValue()+
                " with eta "+timestring(eta));
            pushed_eta = eta.getValue();
            if (ev.booking)
                booking.getValue() && (ev.ETA < eta.getValue());
            else
                booking.getValue() || (ev.ETA < eta.getValue());
        }
    }

    #reasoning method
    cleanUp(String ac){
        logical long atl;
        logical long eta;
        logical boolean b;
        if (runway_info.usingSlot(atl, ac, eta, b)) {
            runway_info.remove(atl.getValue(), ac, eta.getValue(), b.getValue());
        }
    }

    String timestring(logical long x) throws LogicException{
        return DilationController.timeString(x.getValue());
    }
}

```

Figure C.32 RunwayRequest plan

Figure C.33 shows *Takeoff* plan. The *Takeoff* plan handles *TrafficEvent* event and sends *AircraftEvent* event. The *Takeoff* plan implements *body* reasoning methods.

```
import aos.jack.jak.util.timer.DilationController;

/** The Takeoff plan handles a ScheduleRow for a Feeder. It first issues
a booking request for the aircraft concerned. Then it waits until the
aircraft is in the destination airport's control area, at which time it
constructs an Aircraft agent.*/

plan Takeoff extends Plan {

    #handles event TrafficEvent ev;
    static boolean relevant(TrafficEvent ev){
        return ev.mode == TrafficEvent.AIRCRAFT;
    }

    #sends event AircraftEvent landing;
    body(){
        @send(ev.destination,
            landing.request(ev.row.callsign,
                ev.row.booking_ETA,
                ev.row.booking_ETA,
                true));
        System.err.println("Sent "+ev.row);
        @waitFor(afterMillis(ev.row.arrival));
        System.err.println("Arriving "+ev.row);
        new Aircraft(ev.row.callsign, ev.destination, ev.row.arrival_ETA);
    }
}
```

Figure C.33 Takeoff plan

Figure C.34 shows *TakeoffDiscard* plan. The *TakeoffDiscard* plan handles *AircraftEvent* event and implements *relevant* and *body* reasoning methods.

```
/** The TakeoffDiscard plan is a handler for the AircraftEvent.ASSIGN
message returned from the airport for the booking, though it is never
relevant, because we don't care about the on-router behaviour. */

plan TakeoffDiscard extends Plan {

    #handles event AircraftEvent ev;

    static boolean relevant(AircraftEvent ev){
        return false;
    }

    body() {}
}
```

Figure C.34 TakeoffDiscard plan

Figure C.35 shows *Traffic* plan. The *Traffic* plan handles *TrafficEvent* event and reads *Schedule* data. The *Traffic* plan contains *started*, *time* and *callsign* member fields. The *Traffic* plan implements *relevant* and *body* reasoning methods.

```
import java.util.*;
import aos.jack.jak.util.timer.DilationController;

/** The Traffic plan processes the Schedule for a Feeder.*/
plan Traffic extends Plan {

    #handles event TrafficEvent traffic;
    #reads data Schedule schedule;
    static boolean relevant(TrafficEvent traffic) {
        return traffic.mode == TrafficEvent.SCHEDULE;
    }

    Hashtable started = new Hashtable();
    long time = -1;
    String callsign;

    body(){
        String airport = traffic.airport;
        String destination = traffic.destination;

        for (ScheduleRow r = schedule.rows; r != null; r = r.next) {
            @waitFor(afterMillis(r.booking));
            System.err.println("Processing "+r);
            @post(traffic.run(r,airport,destination));
        }
    }
}
```

Figure C.35 Traffic plan

C.5 Code in XML

Figure C.36 shows *Aircraft* agent in XML.

```
<agent id="ag1" name="Aircraft" extends="Agent">
  <hasCapability type="Flying" ref="fly"/>
  <constructor>
    <parameter type="String" ref="id"/>
    <parameter type="String" ref="airport"/>
    <parameter type="long" ref="eta"/>
    <body>
      <![CDATA[
        super(id);
        fly.start(id, airport, eta);
      ]]>
    </body>
  </constructor>
</agent>
```

Figure C.36 Aircraft agent in XML

Figure C.37 shows *Airport* agent in XML.

```
<agent id="ag2" name="Airport" extends="Agent" >
  <import>java.util.Hashtable</import>
  <import>aos.jack.jak.event.TraceMessageEvent</import>
  <hasCapability type="ArrivalSequencing" ref="seq" />
  <constructor>
    <parameter type="String" ref="name"/>
    <parameter type="String []" ref="runway"/>
    <body>
      <![CDATA[
        super(name);
        for (int i = 0; i<runway.length; i++)
          new Runway(runway[i],i);
        seq.enable(runway);
        TracedMessageEvent.tracer.start(this);
      ]]>
    </body>
  </constructor>
</agent>
```

Figure C.37 Airport agent in XML

Figure C.38 shows *Feeder* agent in XML.

```
<agent id="ag3" name="Feeder" extends="Agent" >
  <hasCapability type="TrafficFeeding" ref="feed" />
  <constructor>
    <parameter type="String" ref="name"/>
    <parameter type="String []" ref="destination"/>
    <body>
      <![CDATA[
        super(name);
        feed.load(name,destination);
      ]]>
    </body>
  </constructor>
</agent>
```

Figure C.38 TrafficFeeding agent in XML

Figure C.39 shows *Runway* agent in XML.

```
<agent id="ag4" name="Runway" extends="Agent" >
  <hasCapability type="RunwayAssigning" ref="assign" />
  <constructor>
    <parameter type="String" ref="name"/>
    <parameter type="int" ref="index"/>
    <body>
      <![CDATA[
        super(name);
        assign.setName(name,index);
        TracedMessageEvent.tracer.start(this);
      ]]>
    </body>
  </constructor>
</agent>
```

Figure C.39 Runway agent in XML

Figure C.40 shows *LandingInfo* beliefSet in XML.

```
<beliefSet id="b1" type="LandingInfo" extends="ClosedWorld">
  <field declarationType="value" type="String" name="runway"/>
  <field declarationType="value" type="long" name="ATL"/>
  <linearQuery methodName="get">
    <parameters>
      <parameter type="String" members="logic" ref="runway"/>
      <parameter type="long" members="logic" ref="ATL"/>
    </parameters>
  </linearQuery>
</beliefSet>
```

Figure C.40 LandingInfo beliefSet in XML

Figure C.41 shows *RunwayInfo* beliefSet in XML.

```
<beliefSet id="b2" type="RunwayInfo" extends="ClosedWorld">
  <field declarationType="key" type="long" name="ATL"/>
  <field declarationType="value" type="String" name="aircraft"/>
  <field declarationType="value" type="long" name="ETA"/>
  <field declarationType="value" type="boolean" name="booking"/>
  <field declarationType="normal" type="Stack" name="gui" />
  <method name="setName" returnType="void">
    <parameter type="String" ref="name"/>
    <parameter type="int" ref="index"/>
    <body> <![CDATA[ gui = new Stack(name,index);]]></body>
  </method>
  <indexedQuery methodName="usingSlot">
    <parameters>
      <parameter type="long" members="logic" ref="ATL"/>
      <parameter type="String" members="normal" ref="aircraft"/>
      <parameter type="long" members="logical" ref="ETA"/>
      <parameter type="boolean" members="logical" ref="booking"/>
    </parameters>
  </indexedQuery>
  <indexedQuery methodName="slotUsed">
    <parameters>
      <parameter type="long" members="normal" ref="ATL"/>
      <parameter type="String" members="normal" ref="aircraft"/>
      <parameter type="long" members="normal" ref="ETA"/>
      <parameter type="boolean" members="logical" ref="booking"/>
    </parameters>
  </indexedQuery>
  <method name="newfact" returnType="void">
    <parameter type="Tuple" ref="t"/>
    <parameter type="BeliefState" ref="is"/>
    <parameter type="BeliefState" ref="was"/>
    <body>
      <![CDATA[
        if (gui == null)
          return;
        RunwayInfo__Tuple info = (RunwayInfo__Tuple)t;
        gui.addRow(info.ATL,info.aircraft+ "
["+DilationController.timeString(info.ETA)+"]");
      ]>
    </body>
  </method>
```

```

<parameter type="Tuple" ref="t"/>
<parameter type="BeliefState" ref="was"/>
<body>
  <![CDATA[
    if (gui == null)
      return;
    RunwayInfo__Tuple info = (RunwayInfo__Tuple)t;
    gui.removeRow(info.ATL);
  ]]>
</body>
</method>
</beliefSet>

```

Figure C.41 RunwayInfo beliefSet in XML

Figure C.42 shows *AssignSlot* beliefSet in XML.

```

<plan id="p1" name="AssignSlot" extends="Plan">
  <handlesEvent type="AircraftEvent" ref="ev"/>
  <postsEvent type="WithdrawResponse" ref="response"/>
  <modifiesData type="LandingInfo" ref="landing_info"/>
  <body>
    <![CDATA[ landing_info.add(ev.runway, ev.ATL); ]]>
  </body>
  <relevant>
    <parameter type="Aircraft" ref="ev"/>
    <body>
      <![CDATA[return ev.mode == AircraftEvent.NOTIFY;]]>
    </body>
  </relevant>
</plan>

```

Figure C.42 AssignSlot plan in XML

Figure C.43 shows *FollowApproach* plan in XML.

```

<plan id="p2" name="FollowApproach" extends="Plan">
  <import>import aos.jack.jak.util.timer.DilationController;</import>
  <handlesEvent type="Approaching" ref="ev"/>
  <readsData type="LandingInfo" ref="landing_info"/>
  <context>
    <![CDATA[ landing_info.get(ev.runway, ev.ATL); ]]>
  </context>
  <body>
    <![CDATA[ System.err.println(agent.name()+" following approach to
runway "+ev.runway+" at "+
DilationController.timeString(ev.ATL.getValue()));
    @maintain(landing_info.get(ev.runway, ev.ATL),
    @waitFor(afterMillis(ev.ATL.getValue())));
  ]]>
  </body>
</plan>

```

Figure C.43 FollowApproach plan in XML

Figure C.44 shows InitialApproach plan in XML.

```
<plan id="p3" name="InitialApproach" extends="Plan">
  <handlesEvent type="Approaching" ref="ev"/>
  <postsEvent type="Approaching" ref="land"/>;
  <readsData type="LandingInfo" ref="landing_info"/>
  <context>
    <![CDATA[ landing_info.nFacts() == 0;]]>
  </context>
  <body>
    <![CDATA[
      logical String r;
      logical long a;
      @waitFor(landing_info.get(r,a));
      @achieve(landing_info.get(ev.runway, ev.ATL) &&
               afterMillis(ev.ATL.getValue()),
               land.approach(ev.runway, ev.ATL));
    ]]>
  </body>
</plan>
```

Figure C.44 InitialApproach plan in XML

Figure C.45 shows *MonitorAircraft* plan in XML.

```
<plan id="p4" name="MonitorAircraft" extends="Plan">
  <import>import aos.jack.jak.util.timer.DilationController</import>
  <handlesEvent type="EnterControlArea" ref="ev"/>
  <sendsEvent type="AircraftEvent" ref="ref"/>
  <postsEvent type="Approaching" ref="land"/>
  <readsData type="LandingInfo" ref="landing_info"/>
  <field type="String" members="logical" name="runway"/>
  <field type="long" members="logical" name="ATL"/>
  <body>
    <![CDATA[
      // Confirm landing at arrival ETA
      @send(ev.airport, req.request(ev.id, ev.eta, ev.eta, false));
      @achieve(landing_info.get(runway, ATL) &&
               afterMillis(ATL.getValue()),
               land.approach(runway, ATL));
      System.err.println(ev.id+" landed on runway "+
                          runway.getValue()+" for "+
                          DilationController.timeString(ATL.getValue())+
                          " at "+
                          DilationController.timeString(getAgent().timer.getTime()));
    ]]>
  </body>
</plan>
```

Figure C.45 MonitorAircraft plan in XML

Figure C.46 shows *RequestSlot* plan in XML.

```

<plan id="p5" name="RequestSlot" extends="Plan">
  <import>import aos.jack.util.thread.Semaphore</import>
  <handlesEvent type="AircraftEvent" ref="ev" />
  <relevant>
    <parameter type="AircraftEvent" ref="ev" />
    <body> return ev.mode == AircraftEvent.REQUEST; </body>
  </relevant>
  <usesInterface type="ArrivalSequencing" ref="ev" />
  <modifiesData type="Semaphore" ref="mutex" />
  <parameter type="long" ref="pushed_ETA" value="0" />
  <parameter type="long" ref="ATL" value="-1" />
  <parameter type="String" ref="runway" />
  <body>
    <![CDATA[
      System.err.println("Receieved"+ev);
      if (env.runways == null)
        throw new Error("There are no runways??");
      AircraftEvent query =
ev.request(ev.aircraft, ev.ATL, ev.ETA, ev.booking);
      @waitFor(mutex.planWait());
      System.err.println("Issuing"+ev);
      sendRequests(query);
      receiveReplies(query);
      notifyResult();
      mutex.signal();
    ]]>
  </body>
  <reasoningMethod name="sendRequests" returnType="void">
    <parameter type="AircraftEvent" ref="query" />
    <body>
      <![CDATA[
        for (int i=0; i<env.runways.length; i++)
          @send(env.runways[i], query);
      ]]>
    </body>
  </reasoningMethod>
  <reasoningMethod name="receiveReplies" returnType="void">
    <parameter type="AircraftEvent" ref="query" />
    <body>
      <![CDATA[
        for (int i=0; i<env.runways.length; i++) {
          @waitFor(query.replied());
          AircraftEvent r = (AircraftEvent)query.getReply();
          if (betterSlot(r)) {
            runway = r.from;
            ATL = r.ATL;
            pushed_ETA = r.ETA;
          }
        }
      ]]>
    </body>
  </reasoningMethod>
  <method name="betterSlot" returnType="boolean">
    <body>
      <![CDATA[
        if ((ATL == -1) || (ev.ATL < ATL))
          return true;
      ]]>
    </body>
  </method>
</plan>

```

```

        if (ev.ATL >ATL)
            return false;
        if (ev.ETA == 0)
            return true;
        return (ev.ETA > pushed_ETA);
    ]]>
</body>
</method>
<reasoningMethod name="notifyResult" returnType="void">
    <parameter type="AircraftEvent" ref="query"/>
    <body>
        <![CDATA[
            @send(runway,ev.assign(runway,ATL,ev.aircraft,
                ev.ETA,ev.booking));
            @send(ev.from,ev.confirm(runway,ATL,ev.aircraft));
        ]]>
    </body>
</reasoningMethod>
</plan>

```

Figure C.46 RequestSlot plan in XML

Figure C.47 shows *RunwayAssign* plan in XML.

```

<plan id="p6" name="RunwayAssign" extends="Plan">
    <handlesEvent type="AircraftEvent" ref="ev"/>
    <relevant type="AircraftEvent" ref="ev">
        <body>
            <![CDATA[ return ev.mode == AircraftEvent.ASSIGN; ]]>
        </body>
    </relevant>
    <modifiesData type="RunwayInfo" ref="runway_info"/>
    <body>
        <![CDATA[
            logical String ac;
            logical long eta;
            logical boolean booking;
            if (runway_info.slotUsed(ev.ATL,ac,eta,booking)) {
                System.err.println("PUSH "+ac.getValue()+" by "+ev.aircraft);
                @send(ev.from, ev.request(ac.getValue(),
                    ev.ATL, eta.getValue(),booking.getValue()));}

            runway_info.add(ev.ATL,ev.aircraft,ev.ETA,ev.booking);
            @send(ev.from,ev.confirm(agent.name(),ev.ATL,ev.aircraft));
        ]]>
    </body>
</plan>

```

Figure C.47 RunwayAssign plan in XML

Figure C.48 shows *Takeoff* plan in XML.

```

<plan id="p7" name="Takeoff" extends="Plan">
  <handlesEvent type="TrafficEvent" ref="ev"/>
  <sendsEvent type="AircraftEvent" ref="landing"/>
  <relevant>
    <parameter type="TrafficEvent" name="ev"/>
  </relevant>
  <body>
    <![CDATA[
      @send(ev.destination,
            landing.request(ev.row.callsign,
                           ev.row.booking_ETA,
                           ev.row.booking_ETA,
                           true));

      System.err.println("Sent "+ev.row);
      @waitFor(afterMillis(ev.row.arrival));
      System.err.println("Arriving "+ev.row);
      new
      Aircraft(ev.row.callsign, ev.destination, ev.row.arrival_ETA);
    ]]>
  </body>

```

Figure C.48 Takeoff plan in XML

Figure C.49 shows TakeoffDiscard plan in XML.

```

<plan id="p8" name="TakeoffDiscard" extends="Plan">
  <handlesEvent type="AircraftEvent" ref="ev"/>
  <sendsEvent type="AircraftEvent" ref="landing"/>
  <relevant>
    <parameter type="AircraftEvent" name="ev"/>
  </relevant>
  <body>
    <![CDATA[ ]]>
  </body>
</plan>

```

Figure C.49 TakeoffDiscard plan in XML

The Figure C.50 shows *Traffic* plan in XML.

```

<plan id="p9" name="Traffic" extends="Plan">
  <handlesEvent type="TrafficEvent" ref="traffic"/>
  <sendsEvent type="AircraftEvent" ref="landing"/>
  <readsData type="Schedule" ref="schedule" />
  <relevant>
    <parameter type="TrafficEvent" name="traffic"/>
    <body>
      <![CDATA[
        return traffic.mode == TrafficEvent.SCHEDULE;
      ]]>
    </body>
  </relevant>
  <body><![CDATA[ ]]></body>
</plan>

```

Figure C.50 Traffic plan in XML

The Figure C.51 shows *RunwayRequest* plan in XML.

```
<plan id="p10" name="RunwayRequest" extends="Plan">
  <handlesEvent type="AircraftEvent" ref="ev"/>
  <sendsEvent type="AircraftEvent" ref="landing"/>
  <readsData type="RunwayInfo" ref="runway_info" />
  <relevant>
    <parameter type="AircraftEvent" name="ev"/>
    <body>
      <![CDATA[
        String me = agent.name();
        long t;
        cleanUp(ev.aircraft);
        for (t = env.slotTime(ev.ETA); true; t += env.SLOTGAP) {
          if (freeSlot(t))
            break;
        }
        @reply(ev, ev.assign(me, t, ev.aircraft, pushed_eta, ev.booking));
      ]]>
    </body>
  </relevant>
  <body>
    <![CDATA[
      String me = agent.name();
      long t;
      cleanUp(ev.aircraft);
      for (t = env.slotTime(ev.ETA); true; t += env.SLOTGAP) {
        if (freeSlot(t))
          break;
      }
      @reply(ev, ev.assign(me, t, ev.aircraft, pushed_eta, ev.booking));
    ]]>
  </body>
</plan>
```

Figure C.51 RunwayRequest plan in XML

The Figure C.52 shows *AircraftEvent* event in XML.

```
<event id="ev1" type="AircraftEvent" extends="Event">
  <import>aos.jack.jak.core.*</import>
  <posted methodName="withdraw">
    <![CDATA[ Jak.log.log("Withdraw:withdraw created"); ]]>
  </posted>
</event>
```

Figure C.52 AircraftEvent event in XML

The Figure C.53 shows *Approaching* event in XML.

```

<event id="ev2" type="Approaching" extends="MessageEvent">
  <import>aos.jak.jak.core.Jak</import>
  <field visibility="public" type="account" name="account"/>
  <field visibility="public" type="int" name="pin"/>
  <field visibility="public" type="int" name="amount"/>
  <posted methodName="withdraw">
    <parameter type="int" name="account"/>
    <parameter type="int" name="pin"/>
    <parameter type="int" name="amount"/>
    <![CDATA[
      Jak.log.log("WithdrawRequest:withdraw created");
      this.account = account;
      this.pin = pin;
      this.amount = amount;
      message = "withdraw["+account+", "+pin+"]";
    ]]>
  </posted>
</event>

```

Figure C.53 Approaching event in XML

The Figure C.54 shows *EnterControlArea* event in XML.

```

<event id="ev3" type="EnterControlArea" extends="Event">
  <import>aos.jak.jak.core.Jak</import>
  <field visibility="public" type="boolean" name="approved"/>
  <field visibility="public" type="int" name="balance"/>
  <posted methodName="approval">
    <![CDATA[
      Jak.log.log("WithdrawResponse:approval created");
      this.approved = true;
      this.balance = balance;
      message = "approved";
    ]]>
  </posted>
  <posted methodName="rejection">
    <![CDATA[
      Jak.log.log("WithdrawResponse:rejection created");
      this.approved = false;
      this.balance = 0;
      message = "rejected";
    ]]>
  </posted>
</event>

```

Figure C.54 EnterControlArea event in XML

The Figure C.55 shows *TrafficEvent* event in XML.

```

<event id="ev4" type="TrafficEvent" extends="Event">
  <field type="ScheduleRow" name="row"/>
  <field type="String" name="airport"/>
  <field type="String" name="destination"/>
  <field type="int" name="mode"/>
  <field scope="static" type="int" name="SCHEDULE" value="0" />
  <field scope="static" type="int" name="AIRCRAFT" value="1" />
  <posted methodName="open">
    <parameter type="String" ref="airport"/>
    <parameter type="String" ref="destination"/>
    <![CDATA[
      this.airport = airport;
      this.destination = destination;
      mode = SCHEDULE;
    ]]>
  </posted>
  <posted methodName="run">
    <parameter type="ScheduleRow" ref="row"/>
    <parameter type="String" ref="airport"/>
    <parameter type="String" ref="destination"/>
    <![CDATA[
      this.row = row;
      this.airport = airport;
      this.destination = destination;
      mode = AIRCRAFT;
    ]]>
  </posted>
</event>

```

Figure C.55 TrafficEvent event in XML

The Figure C.56 shows *ArrivalSequencing* capability in XML.

```

<capability id="c1" name="ArrivalSequencing" extends="Capability">
  <handlesEvent type="AircraftEvent"/>
  <privateData type="Semaphore" ref="mutex"/>
  <usesPlan type="RequestSlot"/>
  <field type="String []" name="runways"/>
  <method name="getRunways" returnType="String []">
    <body>
      <![CDATA[
        return runways;
      ]]>
    </body>
  </method>
  <method name="enable" returnType="String []">
    <body>
      <![CDATA[
        this.runways = runways;
        mutex.signal();
      ]]>
    </body>
  </method>
</capability>

```

Figure C.56 ArrivalSequencing capability in XML

The Figure C.57 shows *Flying* capability in XML.

```
<capability id="c2" name="Flying" extends="Capability">
  <privateData type="LandingInfo" ref="landing_info"/>
  <handlesEvent type="AircraftEvent"/>
  <sendsEvent type="AircraftEvent"/>
  <handlesEvent type="EnterControlArea"/>
  <handlesEvent type="Approaching"/>
  <postEvent type="EnterControlArea" ref="enter"/>
  <postEvent type="Approaching" ref="follow"/>
  <usesPlan type="MonitorAircraft"/>
  <usesPlan type="FollowApproach"/>
  <usesPlan type="InitialApproach"/>
  <usesPlan type="AssignSlot"/>
  <method name="start" returnType="void">
    <parameter type="String" ref="id"/>
    <parameter type="String" ref="airport"/>
    <parameter type="long" ref="eta"/>
    <body>
      <![CDATA[
        postEvent(enter.start(id,airport,eta));
      ]]>
    </body>
  </method>
</capability>
```

Figure C.57 Flying capability in XML

The Figure C.58 shows *RunwayAssigning* capability in XML.

```
<capability id="c3" name="RunwayAssigning" extends="Capability">
  <usesPlan type="RunwayRequest"/>
  <usesPlan type="RunwayAssign"/>
  <handlesEvent type="AircraftEvent"/>
  <privateData type="RunwayInfo" ref="runway_info()"/>
  <field type="long" name="SLOTGAP" value="180000"/>
  <method name="slotTime" returnType="long">
    <parameter type="long" ref="time"/>
    <body>
      <![CDATA[
        long x = time/SLOTGAP;
        return (x+1)*SLOTGAP;
      ]]>
    </body>
  </method>
  <method name="setName" returnType="void">
    <parameter type="String" ref="name"/>
    <parameter type="int" ref="index"/>
    <body>
      <![CDATA[ runway_info.setName(name,index); ]]>
    </body>
  </method>
</capability>
```

Figure C.58 RunwayAssigning capability in XML

The Figure C.59 shows *TrafficFeeding* capability in XML

```

<capability id="c4" name="TrafficFeeding" extends="Capability">
  <handlesEvent type="AircraftEvent"/>
  <sendsEvent type="AircraftEvent" ref="request"/>
  <handlesEvent type="TrafficEvent"/>
  <privateData type="Schedule" ref="schedule()"/>
  <usesPlan type="Traffic"/>
  <usesPlan type="Takeoff"/>
  <usesPlan type="TakeoffDiscard"/>
  <field type="TrafficGUI" name="gui"/>
  <postsEvent type="TrafficEvent" ref="traffic"/>
  <method name="load" returnType="void">
    <parameter type="String" ref="name"/>
    <parameter type="String" ref="destination"/>
    <body>
      <![CDATA[
        System.err.println("Feed from "+name+" opened.");
        schedule.load(name+".dat",new TrafficGUI(name));
        postEvent(traffic.open(name,destination));
      ]]>
    </body>
  </method>
</capability>

```

Figure C.59 TrafficFeeding capability in XML

C.6 Evaluation

To evaluate our approach we identified traceability relations manually (see Table C.1) and compared the results with traceability relations identified by the tool (see Table C.2). 31 correct traceability relations had been identified by the tool and 31 traceability relations were missing. The precision and recall calculated were 73.8% and 50%, respectively.

SD Goal	Goal
Allocate Runway Slot	Allocate Runway Slot
Find Best Landing Time for an Aircraft	Landing
Find Best Landing Time for an Aircraft	Find Best Land Time for an Aircraft
SR Goal	Goal
Allocate Runway Slot	Allocate Runway Slot
Find Best Landing Time for an Aircraft	Landing
Find Best Landing Time for an Aircraft	Find Best Land Time for an Aircraft
SR Task	Goal
Query Best Landing Time from All Runway Manager	Query Best Landing Time from All Runway Manager
Query Best Landing Time from All Runway Manager	Landing
Respond Runway Request	Respond Runway Request
Landing	Query Best Landing Time from All Runway Manager
Landing	Landing

Assign Slot	Assign Slot
Initiate Approach	Initiate Aircraft Approach
Follow Approach	Follow Approach Goal
Process Schedule for a Feeder	Process Schedule for a Feeder
Request Booking	Request Booking
PushOut	Push Out
TakeOff	TakeOff Goal
Actor	Agent
Aircraft	Aircraft
Feeder	Feeder
Airport	Airport
Runway	Runway
SD Resource	Message
Slot Allocated	Aircraft Event
ATL	Aircraft Event
SR Resource	Data
Landing Information	landing_info
SD Goal	Agent
Allocate Runway Slot	Airport
Allocate Runway Slot	Runway
Find Best Landing Time for an Aircraft	Aircraft
SR Goal	Agent
Allocate Runway Slot	Airport
Allocate Runway Slot	Runway
Find Best Landing Time for an Aircraft	Aircraft
SR Task	Agent
Query Best Landing Time from All Runway Manager	Airport
Query Best Landing Time from All Runway Manager	Runway
Respond Runway Request	Runway
Landing	Airport
Landing	Runway
Assign Slot	Aircraft
Initiate Approach	Aircraft
Follow Approach	Aircraft
Process Schedule for a Feeder	Feeder
Request Booking	Aircraft
PushOut	Feeder
TakeOff	Feeder
SD Goal	Capability
Allocate Runway Slot	Runway Assigning
Find Best Landing Time for an Aircraft	Flying
SR Goal	Capability
Allocate Runway Slot	Runway Assigning
Find Best Landing Time for an Aircraft	Flying
SR Task	Capability
Query Best Landing Time from All Runway Manager	Arrival Sequencing
Query Best Landing Time from All Runway Manager	Flying
Respond Runway Request	Runway Assigning

Landing	Arrival Sequencing
Landing	Flying
Assign Slot	Flying
Initiate Approach	Flying
Follow Approach	Flying
Process Schedule for a Feeder	Traffic Feeding
Request Booking	Traffic Feeding
PushOut	Traffic Feeding
TakeOff	Traffic Feeding
SR Resource	Plan
Landing Information	Monitor Aircraft
Landing Information	Follow Approach
Landing Information	Initial Approach
Landing Information	Monitor Aircraft
Landing Information	Follow Approach
Landing Information	Assign Slot Plan
Actor	Capability
Aircraft	Flying
Feeder	Traffic Feeding
Airport	Arrival Sequencing
Runway	Runway Assigning
SR Task	Plan
Query Best Landing Time from All Runway Manager	Runway Request
Respond Runway Request	Runway Assign
Landing	Runway Request
Assign Slot	Assign Slot Plan
Initiate Approach	Initial Approach
Follow Approach	Follow Approach
Process Schedule for a Feeder	Traffic
Request Booking	Takeoff
PushOut	Takeoff Discard
TakeOff	Takeoff
SR Goal	Plan
Allocate Runway Slot	Request Slot Plan
Find Best Landing Time for an Aircraft	Monitor Aircraft
SR Resource	Capability
Landing Information	Flying
Landing Information	Flying
SR Resource	Agent
Landing Information	Aircraft
Landing Information	Aircraft

Table C.1 Traceability relations identified manually

Rule ID	Type	SD Goal	Goal
rule1	overlaps	Find Best Landing Time for an Aircraft	Find Best Land Time for an Aircraft
rule1	overlaps	Find Best Landing Time for an Aircraft	Landing
Rule ID	Type	SR Goal	Goal

rule3a	overlaps	Find Best Landing Time for an Aircraft	Find Best Land Time for an Aircraft
rule3a	overlaps	Find Best Landing Time for an Aircraft	Landing
Rule ID	Type	SR Task	Goal
rule4a	overlaps	Landing	Query Best Landing Time from All Runway Manager
rule4a	overlaps	Landing	Progresses an aircraft to Landing
rule4a	overlaps	Landing	Landing
rule4a	overlaps	Assign Slot	Assign Slot
rule4a	overlaps	Initiate Approach	Initiate Aircraft Approach
rule4a	overlaps	Process Schedule for a Feeder	Process Schedule for a Feeder
Rule ID	Type	Actor	Agent
rule49a	overlaps	Aircraft	Aircraft
rule49a	overlaps	Feeder	Feeder
rule49a	overlaps	Airport	Airport
rule49a	overlaps	Runway	Runway
Rule ID	Type	SR Resource	Data
rule52	overlaps	Landing Information	landing_info
Rule ID	Type	SD Goal	Agent
rule12	implements	Find Best Landing Time for an Aircraft	Aircraft
Rule ID	Type	SR Goal	Agent
rule16	implements	Find Best Landing Time for an Aircraft	Aircraft
Rule ID	Type	SR Task	Agent
rule18	implements	Landing	Aircraft
rule18	implements	Landing	Runway
rule18	implements	Assign Slot	Aircraft
rule18	implements	Initiate Approach	Aircraft
rule18	implements	Process Schedule for a Feeder	Feeder
Rule ID	Type	SR Resource	Plan
rule53a	uses	Landing Information	Monitor Aircraft
rule53a	uses	Landing Information	Follow Approach
rule53a	uses	Landing Information	Initial Approach
rule53b	creates	Landing Information	Monitor Aircraft
rule53b	creates	Landing Information	Follow Approach
rule53b	creates	Landing Information	Assign Slot Plan
Rule ID	Type	Actor	Capability
rule56	composedOf	Aircraft	Flying
rule56	composedOf	Feeder	Traffic Feeding
rule56	composedOf	Airport	Arrival Sequencing
rule56	composedOf	Runway	Runway Assigning
Rule ID	Type	SR Task	Plan
rule57	achieves	Landing	Runway Request
rule57	achieves	Landing	Follow Approach
rule57	achieves	Assign Slot	Assign Slot Plan
rule57	achieves	Initiate Approach	Initial Approach
rule57	achieves	Process Schedule for a Feeder	Traffic
Rule ID	Type	SR Goal	Plan
rule58	achieves	Find Best Landing Time for an Aircraft	Monitor Aircraft

Rule ID	Type	SR Resource	Capability
rule54a	uses	Landing Information	Flying
rule54b	creates	Landing Information	Flying
Rule ID	Type	SR Resource	Agent
rule55a	uses	Landing Information	Aircraft
rule55b	creates	Landing Information	Aircraft

Table C.2 Traceability relations identified by the tool

To show how missing elements identified by the tool can assist in the software development process we used the information of missing elements (Table C.3) to complete the models and to fix inconsistencies (e.g. to fix discrepancies between names given by the elements).

Rule ID	SD Goal	Goal
rule1cc	Allocate Runway Slot	
Rule ID	SR Goal	---
rule3cc	Allocate Runway Slot	
Rule ID	SR Plan	Prometheus Goal Prometheus Plan Prometheus Role Prometheus Action
rule4cc	Request Runway	
rule4cc	Respond Runway Request	
rule4cc	Follow Approach	
rule4cc	Request Booking	
rule4cc	TakeOff Discard	
rule4cc	TakeOff	
Rule ID	Prometheus Goal	SD Task SD Goal SR Task SD Goal
rule4cc1	Request Slot	
rule4cc1	Schedule Arrival for a Feeder	
rule4cc1	Assign Runway	
rule4cc1	Push Out	
Rule ID	Goal	Agent
rule12		
Rule ID	SD Resource	Percept Message
rule50cc	Slot Allocated	
rule50cc	ATL	
Rule ID	SD Goal	Agent
rule59cc1	Allocate Runway Slot	
Rule ID	SR Goal	Agent
rule59cc3	Allocate Runway Slot	
Rule ID	SR Task	Agent
rule59cc4	Request Runway	
rule59cc4	Respond Runway Request	
rule59cc4	Follow Approach	
rule59cc4	Request Booking	
rule59cc4	TakeOff Discard	
rule59cc4	TakeOff	
Rule ID	SD Goal	Plan
rule60cc1	Allocate Runway Slot	
Rule ID	SR Goal	Plan

rule60cc3	Allocate Runway Slot	
Rule ID	SR Task	Plan
rule60cc4	Request Runway	
rule60cc4	Respond Runway Request	
rule60cc4	Follow Approach	
rule60cc4	Request Booking	
rule60cc4	TakeOff Discard	
rule60cc4	TakeOff	
Rule ID	SD Goal	Capability
rule60cc1	Allocate Runway Slot	
rule60cc1	Find Best Landing Time for an Aircraft	
Rule ID	SR Goal	Capability
rule60cc3	Allocate Runway Slot	
rule60cc3	Find Best Landing Time for an Aircraft	
Rule ID	SR Task	Capability
rule60cc4	Request Runway	
rule60cc4	Respond Runway Request	
rule60cc4	Landing	
rule60cc4	Assign Slot	
rule60cc4	Initiate Approach	
rule60cc4	Follow Approach	
rule60cc4	Process Schedule for a Feeder	
rule60cc4	Request Booking	
rule60cc4	TakeOff Discard	
rule60cc4	TakeOff	

Table C.3 Missing relations identified by the tool

We run the tool against the new i^* model shown in the Figure C.60 and the new Prometheus model.

The number of correct relations identified was 31, the number of missing elements identified was 31 and the number of wrong relations was 11. The precision and recall calculated was 73,8% and 50% respectively.

Several missing elements were identified by the completeness checking rules. The completeness checking rule *rule1cc* shows that there is a missing traceability relation between Allocate Runway Slot SD Goal and a Prometheus goal (see Table C.4). The rule *rule3cc* shows that there is a missing traceability relation between Allocate Runway Slot SR Goal and a Prometheus goal (see Table C.5).

The rule *rule4cc1* shows that any relation between Request Slot Prometheus goal and a SD Task, or SD Goal, or SR Task, or SR Goal was identified (see Table C.4 and Table C.5). The

action taken to correct this discrepancy was to rename the name of Request Slot goal in the Prometheus model to Allocate Runway Slot.

Rule ID	SD Goal	Goal
rule1cc	Allocate Runway Slot	

Table C.4 Missing relations between SD Goal and Prometheus Goal

Rule ID	SR Goal	Goal
Rule3cc	Allocate Runway Slot	

Table C.5 Missing relations between SR Goal and Prometheus Goal

We changed the name of Request Slot to Allocate Runway Slot to fix the discrepancy between names and the relation was identified.

Rule rule4cc shows that there are missing traceability relations between Request Runway, Respond Runway Request, Follow Approach, Request Booking, TakeOff Discard, TakeOff SR Tasks and Prometheus Goals (see Table C.3). We added Request Booking and TakeOff Goal goals in the Prometheus to complete the model. After analysing the models, we identified that there is a missing relation between Follow Approach SR Task in i^* and Progress an aircraft to Landing in Prometheus model. To fix the discrepancy between the names given, we changed the name in the Prometheus model from Progress an aircraft to Landing to Follow Approach Goal. The Figure C.60 and Figure C.61 show i^* model and Prometheus goal diagram updated.

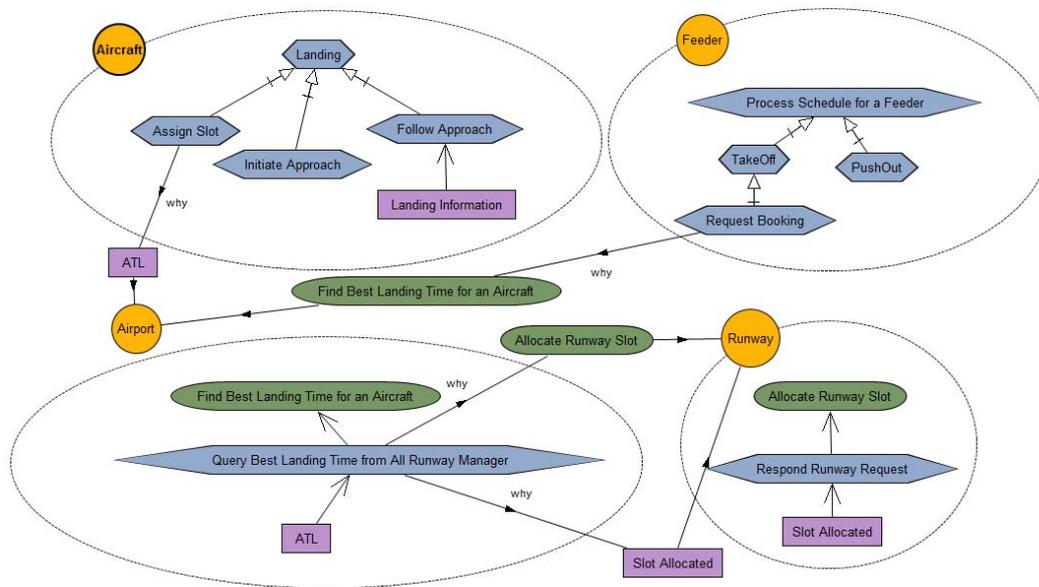


Figure C.60 Air Traffic Control Environment *i** model version 1



Figure C.61 Prometheus goal diagram

Rule ID	SR Task	Prometheus Goal
rule4cc	Request Runway	
rule4cc	Respond Runway Request	
rule4cc	Follow Approach	
rule4cc	Request Booking	
rule4cc	TakeOff Discard	
rule4cc	TakeOff	

Table C.6 Missing relations between SR Plan and Prometheus Goal

The rule rule4cc1 shows that there are missing traceability relations between Request Slot, Schedule Arrival for a Feeder, Assign Runway, Push Out goals in Prometheus and SD Task, or SD Goal, or SR Task, or SR Goal (see Table C.7).

Rule ID	Prometheus Goal	SD Task SD Goal SR Task SD Goal
rule4cc1	Request Slot	
rule4cc1	Schedule Arrival for a Feeder	
rule4cc1	Assign Runway	
rule4cc1	Push Out	

Table C.7 Missing relations between Prometheus Goal and SD/SR Task or SD/SR Goal

There is a discrepancy between the names given to the TakeOff Discard SR Task and Push Out goal in Prometheus. We changed the name of the SR Task from TakeOff Discard to Push Out in the i^* model. There is also a discrepancy between the names given to Respond Runway Request SR Task and Assign Runway in Prometheus. We changed the name of the Assign Runway goal in Prometheus to Respond Runway Request. No traceability relation was found between Schedule Arrival for a Feeder Prometheus goal and a SD Task, or SD Goal, or SR Task, or SR Goal. We decide to remove it from the Prometheus model. The Schedule Arrival for a Feeder is similar to the Process Schedule for a Feeder goal (another action could be to add a SR goal in the in i^* model that would have a means-end relationship with the Process Schedule for a Feeder). The traceability relation between Request Runway SR Task in i^* and Query Best Landing Time from All Runway Manager was not identified by the tool. In order to fix this discrepancy between names we changed the name from Runway Request in the i^* model to Query Best Landing Time from All Runway Manager.

The rule rule50cc shows that any traceability relation was identified between Slot Allocated and ATL SD Resource in i^* and a Percept or a Message in Prometheus (see Table C.8). We identified that the carried information ATL and Slot Allocated for the Aircraft Event message was missing. We fix the incompleteness adding ATL and Slot Allocated to the carried information property of Aircraft Event message.

Rule ID	SD Resource	Percept Message
rule50cc	Slot Allocated	
rule50cc	ATL	

Table C.8 Missing relations between SD Resource and Prometheus Percept

Rule rule59cc1 shows that any traceability relation was found between Allocate Runway Slot SD goal and a Prometheus agent (see Table C.9) and the rule rule59cc3 shows that any traceability relation was identified between Allocate Runway Slot SR goal and a Prometheus agent (see Table C.10). We added to the list of goals achieved by the Runway Prometheus agent the Allocate Runway Slot (named before by Request Slot).

Rule ID	SD Goal	Agent
rule59cc1	Allocate Runway Slot	

Table C.9 Missing relations between SD Goal and a Prometheus Agent

Rule ID	SR Goal	Agent
rule59cc3	Allocate Runway Slot	

Table C.10 Missing relation between a SR Goal and an Agent

Table C.11 shows missing relations between Request Runway, Respond Runway Request, Follow Approach, Request Booking, TakeOff Discard, and Take Off SR Tasks and agents in Prometheus. We added Query Best Landing Time from All Runway Manager (before named as Request Runway) to the list of goals achieved by Airport Prometheus Agent. No action was necessary for Request Runway Request and Follow Approach. The goal name changed from Assign Runway to Respond Runway Request and from Progresses an aircraft to Landing to Follow Approach resolved the incompleteness. We added Request Booking to the list of goals achieved by Aircraft Prometheus Agent and TakeOff and Push Out to the list of goals achieved by the Feeder Prometheus Agent.

Rule ID	SR Task	Agent
rule59cc4	Request Runway	
rule59cc4	Respond Runway Request	
rule59cc4	Follow Approach	
rule59cc4	Request Booking	
rule59cc4	TakeOff Discard	
rule59cc4	TakeOff	

Table C.11 Missing relations between a SR Task and an Agent

Table C.12 shows missing relations between Allocate Runway Slot SD Goal and plans in Prometheus and the table C.13 shows missing relations between Allocate Runway Slot SR goal

and a plan in Prometheus. Any action was necessary because the change of the Request Slot goal name to Allocate Runway Slot fixed the incompleteness.

Rule ID	SD Goal	Plan
rule60cc1	Allocate Runway Slot	

Table C.12 Missing relations between a SD Goal and a Prometheus Plan

Rule ID	SR Goal	Plan
rule60cc3	Allocate Runway Slot	

Table C.13 Missing relations between a SR Goal and a Prometheus Plan

Table C.14 shows missing relations between Request Runway, Respond Runway Request, Follow Approach, Request Booking, TakeOff Discard, and Take Off SR Tasks and plans in Prometheus. We added Query Best Landing Time from All Runway Manager (before named as Request Runway) to the list of goals achieved by Airport. No action was necessary for Request Runway Request and Follow Approach. The goal name changed from Assign Runway to Respond Runway Request and from Progresses an aircraft to Landing to Follow Approach resolved the incompleteness. We added Request Booking and TakeOff to the list of goals achieved by TakeOff Prometheus Agent. We added Push Out to the list of goals achieved by the Takeoff Discard Prometheus Plan.

Rule ID	SR Task	Plan
rule60cc4	Request Runway	
rule60cc4	Respond Runway Request	
rule60cc4	Follow Approach	
rule60cc4	Request Booking	
rule60cc4	TakeOff Discard	
rule60cc4	TakeOff	

Table C.14 Missing relations between a SR Task and Prometheus Plan

Table C.15 shows missing relations between Allocate Runway Slot and Find Best Landing Time for an Aircraft SD goals and Prometheus capabilities. Table C.16 shows missing relations between Allocate Runway Slot and Find Best Landing Time for an Aircraft SR goals and capabilities in Prometheus. We added Allocate Runway Slot goal to the list of goals achieved by Runway Assigning and Find Best Landing Time for an Aircraft to the list of goals achieved by Arrival Sequencing.

Rule ID	SD Goal	Capability
rule60cc1	Allocate Runway Slot	
rule60cc1	Find Best Landing Time for an Aircraft	

Table C.15 Missing links between a SD Goal and Prometheus Capability

Rule ID	SR Goal	Capability
rule60cc3	Allocate Runway Slot	
rule60cc3	Find Best Landing Time for an Aircraft	

Table C.16 Missing links between a SR Goal and Prometheus Capability

Table C.17 shows missing relations between Request Runway, Respond Runway Request, Landing, Assign Slot, Initiate Approach, Follow Approach, Process Schedule for a Feeder, Request Booking, TakeOff Discard, TakeOff SR tasks in i^* and Prometheus Capabilities. We added Query Best Landing Time from All Runway Manager to the list of goals achieved by Arrival Sequencing capability. We added Respond Runway Request to the list of goals achieved by Runway Assigning. We added Landing, Assign Slot, Initiate Aircraft Approach, and Follow Approach to the list of goals achieved by Flying capability. We added Process Schedule for a Feeder, Request Booking, Push Out and Take Off Goal to the list of goals achieved by the Feeder capability.

Rule ID	SR Task	Capability
rule60cc4	Request Runway	
rule60cc4	Respond Runway Request	
rule60cc4	Landing	
rule60cc4	Assign Slot	
rule60cc4	Initiate Approach	
rule60cc4	Follow Approach	
rule60cc4	Process Schedule for a Feeder	
rule60cc4	Request Booking	
rule60cc4	TakeOff Discard	
rule60cc4	TakeOff	

Table C.17 Missing relations between a SR Task and Prometheus Capability

After to complete the model and run the traceability tool again the number of correct relations identified was 65, the number of missing elements identified was 4 and the number of wrong relations was 18. The precision and recall calculated was 78,82% and 94,36%, respectively.

Rule ID	SD Goal	Goal
rule1	Allocate Runway Slot	Allocate Runway Slot
rule1	Find Best Landing Time for an Aircraft	Landing
rule1	Find Best Landing Time for an Aircraft	Find Best Land Time for an Aircraft
Rule ID	SR Goal	Goal
rule3a	Allocate Runway Slot	Allocate Runway Slot
rule3a	Find Best Landing Time for an Aircraft	Landing
rule3a	Find Best Landing Time for an Aircraft	Find Best Land Time for an Aircraft
Rule ID	SR Task	Goal
rule4a	Query Best Landing Time from All Runway Manager	Query Best Landing Time from All Runway Manager
rule4a	Query Best Landing Time from All Runway Manager	Landing
rule4a	Respond Runway Request	Respond Runway Request
rule4a	Landing	Query Best Landing Time from All Runway Manager
rule4a	Landing	Landing
rule4a	Assign Slot	Assign Slot
rule4a	Initiate Approach	Initiate Aircraft Approach
rule4a	Follow Approach	Follow Approach Goal
rule4a	Process Schedule for a Feeder	Process Schedule for a Feeder
rule4a	Request Booking	Request Booking
rule4a	PushOut	Push Out
rule4a	TakeOff	TakeOff Goal
Rule ID	Actor	Agent
rule49a	Aircraft	Aircraft
rule49a	Feeder	Feeder
rule49a	Airport	Airport
rule49a	Runway	Runway
Rule ID	SD Resource	Message
rule50	Slot Allocated	Aircraft Event
rule50	ATL	Aircraft Event
Rule ID	SR Resource	Data
rule52	Landing Information	landing_info
Rule ID	SD Goal	Agent
rule12	Allocate Runway Slot	Airport
rule12	Allocate Runway Slot	Runway
rule12	Find Best Landing Time for an Aircraft	Aircraft
Rule ID	SR Goal	Agent
rule16	Allocate Runway Slot	Airport
rule16	Allocate Runway Slot	Runway
rule16	Find Best Landing Time for an Aircraft	Aircraft
Rule ID	SR Task	Agent
rule18	Query Best Landing Time from All Runway Manager	Airport
rule18	Query Best Landing Time from All	Runway

	Runway Manager	
rule18	Respond Runway Request	Runway
rule18	Landing	Airport
rule18	Landing	Runway
rule18	Assign Slot	Aircraft
rule18	Initiate Approach	Aircraft
rule18	Follow Approach	Aircraft
rule18	Process Schedule for a Feeder	Feeder
rule18	Request Booking	Aircraft
rule18	PushOut	Feeder
rule18	TakeOff	Feeder
Rule ID	SD Goal	Capability
rule19	Allocate Runway Slot	Runway Assigning
rule19	Find Best Landing Time for an Aircraft	Flying
Rule ID	SR Goal	Capability
rule23	Allocate Runway Slot	Runway Assigning
rule23	Find Best Landing Time for an Aircraft	Flying
Rule ID	SR Task	Capability
rule25	Query Best Landing Time from All Runway Manager	Arrival Sequencing
rule25	Query Best Landing Time from All Runway Manager	Flying
rule25	Respond Runway Request	Runway Assigning
rule25	Landing	Arrival Sequencing
rule25	Landing	Flying
rule25	Assign Slot	Flying
rule25	Initiate Approach	Flying
rule25	Follow Approach	Flying
rule25	Process Schedule for a Feeder	Traffic Feeding
rule25	Request Booking	Traffic Feeding
rule25	PushOut	Traffic Feeding
rule25	TakeOff	Traffic Feeding
Rule ID	SR Resource	Plan
rule53a	Landing Information	Monitor Aircraft
rule53a	Landing Information	Follow Approach
rule53a	Landing Information	Initial Approach
rule53b	Landing Information	Monitor Aircraft
rule53b	Landing Information	Follow Approach
rule53b	Landing Information	Assign Slot Plan
Rule ID	Actor	Capability
rule56	Aircraft	Flying
rule56	Feeder	Traffic Feeding
rule56	Airport	Arrival Sequencing
rule56	Runway	Runway Assigning
Rule ID	SR Task	Plan
rule57	Query Best Landing Time from All Runway Manager	Runway Request
rule57	Respond Runway Request	Runway Assign
rule57	Landing	Runway Request
rule57	Assign Slot	Assign Slot Plan
rule57	Initiate Approach	Initial Approach

rule57	Follow Approach	Follow Approach
rule57	Process Schedule for a Feeder	Traffic
rule57	Request Booking	Takeoff
rule57	PushOut	Takeoff Discard
rule57	TakeOff	Takeoff
Rule ID	SR Goal	Plan
rule58	Allocate Runway Slot	Request Slot Plan
rule58	Find Best Landing Time for an Aircraft	Monitor Aircraft
Rule ID	SR Resource	Capability
rule54a	Landing Information	Flying
rule54b	Landing Information	Flying
Rule ID	SR Resource	Agent
rule55a	Landing Information	Aircraft
rule55b	Landing Information	Aircraft

Table C.18 Traceability relations between i^* and Prometheus

Rule ID	JACK BeliefSet	Prometheus Data
RulePJ2cc1	LandingInfo	
RulePJ2cc1	RunwayInfo	
Rule ID	Prometheus Goal	JACK Agent
RulePJ5cc1	Request Slot	
RulePJ5cc1	Process Schedule for a Feeder	
RulePJ5cc1	Schedule Arrival for a Feeder	
RulePJ5cc1	Query Best Landing Time from All Runway Manager	
RulePJ5cc1	Assign Runway	
RulePJ5cc1	Find Best Land Time for an Aircraft	
RulePJ5cc1	Push Out	
RulePJ5cc1	Progresses an aircraft to Landing	
RulePJ5cc1	Initiate Aircraft Approach	
RulePJ5cc1	Assign Slot	
RulePJ5cc1	Landing	
Rule ID	Prometheus Message	JACK Agent
RulePJ12cc1	Traffic Event	
RulePJ12cc1	Enter Control Area	

Table C.19 Missing relations between JACK and Prometheus

Appendix D – Electronic Bookstore Case Study

This document describes the development of a multi-agent system to implement the Electronic Bookstore (EB) used as a case study to evaluate our approach to generate traceability relations automatically and to identify missing elements between artefacts created during the development of a multi-agent system.

D.1 JACK Agent vs Prometheus Goal

Table D.1 shows the traceability relations identified manually between Prometheus Goal and JACK Agent and the table D.2 shows the traceability relations identified by the tool between Prometheus Goal and JACK Agent identified by the tool. Relations between a Prometheus Goal and a JACK Agent found by the tool are based on previous relations identified between Prometheus Agents and JACK Agents and relations defined in the Prometheus model between Prometheus Goals and Prometheus Agents. The number of relations identified manually is 38, the number of relations identified by the tool is 38, and the number of correct relations is 38. The precision and recall calculated is 100%.

The reason why precision and recall are equal to 100% is because the tool had identified previously all the relations between Prometheus Agents and JACK Agents completely and correctly and all the relations between Prometheus Goals and Prometheus Agents had been completed defined.

Prometheus Goal	JACK Agent
Retrieve Item Details	Stock Manager
Register New Customer	Customer Relations
Calculate Delivery Time and Price	Delivery Manager
Check Availability	Stock Manager
Perform Keyword Search	Stock Manager
Perform Advanced Search	Stock Manager
Find BestSellers	Stock Manager
Find Book Details	Stock Manager
Find Books by Category	Stock Manager
Find New Releases	Stock Manager
Find Special Offers	Stock Manager
Find Categories	Stock Manager
Find Top Ten BestSellers	Stock Manager
Show New Customer Page	Sales Assistant
Show Updated Basket	Sales Assistant
Show Delivery Time and Price Options	Sales Assistant
Show Top Ten BestSellers	Sales Assistant
Show Order Confirmation	Sales Assistant

Show Account Details Page	Sales Assistant
Proceed To CheckOut	Sales Assistant
Update Basket	Sales Assistant
Get Customer Details	Customer Relations
Show User Details	Sales Assistant
Show Advanced Search Form	Sales Assistant
Show Advanced Search Result	Sales Assistant
Show Basket Page	Sales Assistant
Show Best Sellers Page	Sales Assistant
Show Book Details Page	Sales Assistant
Show Books by Category Page	Sales Assistant
Show Top Ten BestSellers	Sales Assistant
Show Contact Information	Sales Assistant
Show Help Information	Sales Assistant
Show Keyword Search Result	Sales Assistant
Show New Releases	Sales Assistant
Show Password Assistance	Sales Assistant
Show LogIn Page	Sales Assistant
Show Special Offers Page	Sales Assistant
Show Categories Page	Sales Assistant
Validate User	Customer Relations
Process Payment	Payment Processor

Table D.1 Relations identified manually between Prometheus Goal and JACK Agent

Rule ID	Prometheus Goal	JACK Agent
rulePJ5a	Show Contact Information	SalesAssistant
rulePJ5a	Show Help Information	SalesAssistant
rulePJ5a	Show User Details	SalesAssistant
rulePJ5a	Show Account Details Page	SalesAssistant
rulePJ5a	Show Advanced Search Result	SalesAssistant
rulePJ5a	Show Advanced Search Form	SalesAssistant
rulePJ5a	Show Books Details Page	SalesAssistant
rulePJ5a	Show Books by Category Page	SalesAssistant
rulePJ5a	Show LogIn Page	SalesAssistant
rulePJ5a	Proceed To CheckOut	SalesAssistant
rulePJ5a	Show Categories Page	SalesAssistant
rulePJ5a	Show Top Ten BestSellers	SalesAssistant
rulePJ5a	Show Order Confirmation	SalesAssistant
rulePJ5a	Show Delivery Time and Price Options	SalesAssistant
rulePJ5a	Show Keyword Search Result	SalesAssistant
rulePJ5a	Show Updated Basket	SalesAssistant
rulePJ5a	Show Best Sellers Page	SalesAssistant
rulePJ5a	Show New Releases	SalesAssistant
rulePJ5a	Show Special Offers Page	SalesAssistant
rulePJ5a	Update Basket	SalesAssistant
rulePJ5a	Show Basket Page	SalesAssistant
rulePJ5a	Show New Customer Page	SalesAssistant
rulePJ5a	Get Customer Details	CustomerRelations
rulePJ5a	Validate User	CustomerRelations
rulePJ5a	Register New Customer	CustomerRelations
rulePJ5a	Calculate Delivery Time and Price	DeliveryManager
rulePJ5a	Process Payment	PaymentProcessor
rulePJ5a	Find Books by Category	StockManager
rulePJ5a	Find Top Ten BestSellers	StockManager
rulePJ5a	Perform Keyword Search	StockManager
rulePJ5a	Perform Advanced Search	StockManager

rulePJ5a	Retrieve Item Details	StockManager
rulePJ5a	Check Availability	StockManager
rulePJ5a	Find BestSellers	StockManager
rulePJ5a	Find Book Details	StockManager
rulePJ5a	Find New Releases	StockManager
rulePJ5a	Find Special Offers	StockManager
rulePJ5a	Find Categories	StockManager

Table D.2 Relations identified by the tool between Prometheus Goal and JACK Agent

D.2 JACK Agent vs Prometheus Role

Table D.3 shows the traceability relations identified manually between Prometheus Role and JACK Agent and table D.4 shows the traceability relations identified by the tool between Prometheus Role and JACK Agent. Relations between a Prometheus Role and a JACK Agent found by the tool are based on previous relations identified between Prometheus Agents and JACK Agents and relations defined in the Prometheus model between Prometheus Role and Prometheus Agents. The number of relations identified manually is 9, and the number of relations identified by the tool is 9. The precision and recall calculated is 100%.

The reason why precision and recall are equal to 100% is because the tool had identified previously all the relations between Prometheus Agents and JACK Agents completely and correctly and all the relations between Prometheus Roles and Prometheus Agents had been completed defined.

Prometheus Role	JACK Agent
Customer Management	SalesAssistant
Order Management	SalesAssistant
Product Information Management	SalesAssistant
Shop Information Management	SalesAssistant
Customer Relationship Management	CustomerRelations
Stock Management	StockManager
Search Management	StockManager
Service Delivery Management	DeliveryManager
Payment Management	PaymentProcessor

Table D.3 Relations identified manually between Prometheus Role and JACK Agent

Rule ID	Prometheus Role	JACK Agent
rulePJ6a	Shop Information Management	SalesAssistant
rulePJ6a	Product Information Management	SalesAssistant
rulePJ6a	Order Management	SalesAssistant
rulePJ6a	Customer Management	SalesAssistant
rulePJ6a	Customer Relationship	CustomerRelations

	Management	
rulePJ6a	Service Delivery Management	DeliveryManager
rulePJ6a	Payment Management	PaymentProcessor
rulePJ6a	Stock Management	StockManager
rulePJ6a	Search Management	StockManager

Table D.4 Relations identified by the tool between Prometheus Role and JACK Agent

D.3 JACK Agent vs Prometheus Agent

Table D.5 shows the traceability relations identified manually between Prometheus Role and JACK Agent and table D.6 shows the traceability relations identified by the tool between Prometheus Role and JACK Agent. The number of relations identified manually is 5, and the number of relations identified by the tool is 5. The precision and recall calculated is 100%.

Prometheus Agent	JACK Agent
Customer Relations	CustomerRelations
Delivery Manager	DeliveryManager
Payment Processor	PaymentProcessor
Sales Assistant	SalesAssistant
Stock Manager	StockManager
	DispatcherAgent

Table D.5 Relations identified manually between Prometheus Agent and JACK Agent

Rule ID	Prometheus Agent	JACK Agent
rulePJ4a	Sales Assistant	SalesAssistant
rulePJ4a	Customer Relations	CustomerRelations
rulePJ4a	Delivery Manager	DeliveryManager
rulePJ4a	Payment Processor	PaymentProcessor
rulePJ4a	Stock Manager	StockManager

Table D.6 Relations identified by the tool between Prometheus Agent and JACK Agent

D.4 JACK Agent vs Prometheus Capability

Table D.7 shows the traceability relations identified manually between Prometheus Capability and JACK Agent and table D.8 shows the traceability relations identified by the tool between Prometheus Capability and JACK Agent. The number of relations identified manually is 39, and the number of relations identified by the tool is 39. The precision and recall calculated is 100%.

Prometheus Capability	JACK Agent
Add Customer Capability	CustomerRelations
Add Customer Response Capability	SalesAssistant
Add Item to Basket Capability	StockManager
Add Item To Basket Response Capability	SalesAssistant
Advanced Search Capability	StockManager
Advanced Search Response Capability	SalesAssistant
Calculate Delivery Time and Price Capability	DeliveryManager
Check Stock Capability	StockManager
Find BestSellers Capability	StockManager
Find Book Details Capability	StockManager
Find Books by Category Capability	StockManager
Find New Releases Capability	StockManager
Find Special Offers Capability	StockManager
Find Subjects Category Capability	StockManager
Find Top Ten BestSellers Capability	StockManager
Get Delivery Options Response Capability	SalesAssistant
Keyword Search Capability	StockManager
Keyword Search Response Capability	SalesAssistant
Log Out Response Capability	SalesAssistant
LogIn Response Capability	SalesAssistant
Place Order Response Capability	SalesAssistant
Proceed To Check Out Response Capability	SalesAssistant
Retrieve Customer Details Capability	CustomerRelations
Show Account Detail Response Capability	SalesAssistant
Show Advanced Search Form Response Capability	SalesAssistant
Show Basket Response Capability	SalesAssistant
Show BestSellers Response Capability	SalesAssistant
Show Book Details Response Capability	SalesAssistant
Show Books by CategoryResponse Capability	SalesAssistant
Show Bookstore Main Page Response Capability	SalesAssistant
Show Contact Information Response Capability	SalesAssistant
Show Help Information Response Capability	SalesAssistant
Show New Releases Response Capability	SalesAssistant
Show SignIn Form Response Capability	SalesAssistant
Show Special Offers Response Capability	SalesAssistant
Show Subjects Response Capability	SalesAssistant
SignIn Capability	CustomerRelations
Update Basket Response Capability	SalesAssistant
Validate Credit Card Capability	Payment Processor

Table D.7 Relations identified manually Prometheus Capability and JACK Agent

Rule ID	Prometheus Capability	JACK Agent
rulePJ8a	Add Customer Response Capability	SalesAssistant
rulePJ8a	Get Delivery Options Response Capability	SalesAssistant
rulePJ8a	LogIn Response Capability	SalesAssistant
rulePJ8a	Place Order Response Capability	SalesAssistant
rulePJ8a	Add Item To Basket Response Capability	SalesAssistant
rulePJ8a	Advanced Search Response Capability	SalesAssistant
rulePJ8a	Keyword Search Response Capability	SalesAssistant
rulePJ8a	Show Books by CategoryResponse Capability	SalesAssistant
rulePJ8a	Show Book Details Response Capability	SalesAssistant
rulePJ8a	Log Out Response Capability	SalesAssistant
rulePJ8a	Show Bookstore Main Page Response Capability	SalesAssistant
rulePJ8a	Show BestSellers Response Capability	SalesAssistant
rulePJ8a	Show New Releases Response Capability	SalesAssistant
rulePJ8a	Show Special Offers Response Capability	SalesAssistant
rulePJ8a	Show Subjects Response Capability	SalesAssistant
rulePJ8a	Proceed To Check Out Response Capability	SalesAssistant
rulePJ8a	Show Account Detail Response Capability	SalesAssistant
rulePJ8a	Show Advanced Search Form Response Capability	SalesAssistant
rulePJ8a	Show Basket Response Capability	SalesAssistant
rulePJ8a	Show Contact Information Response Capability	SalesAssistant
rulePJ8a	Show Help Information Response Capability	SalesAssistant
rulePJ8a	Show SignIn Form Response Capability	SalesAssistant
rulePJ8a	Update Basket Response Capability	SalesAssistant
rulePJ8a	Add Customer Capability	CustomerRelations
rulePJ8a	Retrieve Customer Details Capability	CustomerRelations
rulePJ8a	SignIn Capability	CustomerRelations
rulePJ8a	Calculate Delivery Time and Price Capability	DeliveryManager
rulePJ8a	Validate Credit Card Capability	PaymentProcessor
rulePJ8a	Check Stock Capability	StockManager
rulePJ8a	Add Item to Basket Capability	StockManager
rulePJ8a	Advanced Search Capability	StockManager
rulePJ8a	Keyword Search Capability	StockManager
rulePJ8a	Find Books by Category Capabability	StockManager
rulePJ8a	Find Book Details Capability	StockManager
rulePJ8a	Find Top Ten BestSellers Capability	StockManager
rulePJ8a	Find BestSellers Capability	StockManager
rulePJ8a	Find New Releases Capability	StockManager
rulePJ8a	Find Special Offers Capability	StockManager
rulePJ8a	Find Subjects Category Capability	StockManager

Table D.8 Relations identified by the tool between Prometheus Capability and JACK Agent

D.5 JACK Agent vs Prometheus Plan

Table D.9 shows the traceability relations identified manually between Prometheus Plan and JACK Agent and table D.10 shows the traceability relations identified by the tool between Prometheus Plan and JACK Agent. The number of relations identified manually is 39, and the number of relations identified by the tool is 39. Precision and recall calculated is 100%.

Prometheus Plan	JACK Agent
Add Item to Basket Plan	StockManager
Add New Customer	CustomerRelations
Calculate Delivery Time and Price Plan	DeliveryManager
Check Stock	StockManager
Execute Advanced Search	StockManager
Execute Keyword Search	StockManager
Find BestSellers Plan	StockManager
Find Book Details Plan	StockManager
Find Books by Category Plan	StockManager
Find New Releases Plan	StockManager
Find Special Offers Plan	StockManager
Find Subjects	StockManager
Find Top Ten BestSellers Plan	StockManager
Respond Add Customer Request	SalesAssistant
Respond Add Item to Basket Request	SalesAssistant
Respond Get Delivery Options Request	SalesAssistant
Respond Log Out Request	SalesAssistant
Respond Place Order Request	SalesAssistant
Respond SignIn Request	SalesAssistant
Respond to Proceed to CheckOut Request	SalesAssistant
Respond Update Basket Request	SalesAssistant
Retrieve Customer Details	CustomerRelations
Show Account	SalesAssistant
Show Advanced Search Form Plan	SalesAssistant
Show Advanced Search Result Plan	SalesAssistant
Show Basket Plan	SalesAssistant
Show BestSellers Plan	SalesAssistant
Show Book Details	SalesAssistant
Show Books by Category Plan	SalesAssistant
Show Bookstore Main PagePlan	SalesAssistant
Show Contact Information Plan	SalesAssistant
Show Help Information Plan	SalesAssistant
Show Keyword Search Result Plan	SalesAssistant
Show New Releases Plan	SalesAssistant
Show SignIn form	SalesAssistant
Show Special Offers Plan	SalesAssistant
Show Subjects Plan	SalesAssistant
Sign In	CustomerRelations
Validate Credit Card	PaymentProcessor

Table D.9 Relations identified manually between Prometheus Plan and JACK Agent

Rule ID	Prometheus Plan	JACK Agent
rulePJ9a	Respond Add Customer Request	SalesAssistant
rulePJ9a	Respond Get Delivery Options Request	SalesAssistant
rulePJ9a	Respond SignIn Request	SalesAssistant
rulePJ9a	Respond Place Order Request	SalesAssistant
rulePJ9a	Respond Add Item to Basket Request	SalesAssistant
rulePJ9a	Show Advanced Search Result	SalesAssistant

	Plan	
rulePJ9a	Show Keyword Search Result Plan	SalesAssistant
rulePJ9a	Show Books by Category Plan	SalesAssistant
rulePJ9a	Show Book Details	SalesAssistant
rulePJ9a	Respond Log Out Request	SalesAssistant
rulePJ9a	Show Bookstore Main Page Plan	SalesAssistant
rulePJ9a	Show BestSellers Plan	SalesAssistant
rulePJ9a	Show New Releases Plan	SalesAssistant
rulePJ9a	Show Special Offers Plan	SalesAssistant
rulePJ9a	Show Subjects Plan	SalesAssistant
rulePJ9a	Respond to Proceed to CheckOut Request	SalesAssistant
rulePJ9a	Show Account	SalesAssistant
rulePJ9a	Show Advanced Search Form Plan	SalesAssistant
rulePJ9a	Show Basket Plan	SalesAssistant
rulePJ9a	Show Contact Information Plan	SalesAssistant
rulePJ9a	Show Help Information Plan	SalesAssistant
rulePJ9a	Show SignIn Form	SalesAssistant
rulePJ9a	Respond Update Basket Request	SalesAssistant
rulePJ9a	Add New Customer	CustomerRelations
rulePJ9a	Retrieve Customer Details	CustomerRelations
rulePJ9a	Sign In	CustomerRelations
rulePJ9a	Calculate Delivery Time and Price Plan	DeliveryManager
rulePJ9a	Validate Credit Card	PaymentProcessor
rulePJ9a	Check Stock	StockManager
rulePJ9a	Add Item to Basket Plan	StockManager
rulePJ9a	Execute Advanced Search	StockManager
rulePJ9a	Execute Keyword Search	StockManager
rulePJ9a	Find Books by Category Plan	StockManager
rulePJ9a	Find Book Details Plan	StockManager
rulePJ9a	Find Top Ten BestSellers Plan	StockManager
rulePJ9a	Find BestSellers Plan	StockManager
rulePJ9a	Find New Releases Plan	StockManager
rulePJ9a	Find Special Offers Plan	StockManager
rulePJ9a	Find Subjects	StockManager

Table D.10 Relations identified by the tool between Prometheus Plan and JACK Agent

D.6 JACK Agent vs Prometheus Percept

Table D.11 shows the traceability relations identified manually between Prometheus Plan and JACK Agent and table D.12 shows the traceability relations identified by the tool between Prometheus Plan and JACK Agent. The number of relations identified manually is 24, and the number of relations identified by the tool is 15. Precision and recall calculated is 100% and 62.50%, respectively. The reason why recall is low it is because the rule rulePJ10a only capture

Percepts defined on the System Overview Diagram. Table D.13 shows the information about missing traceability relations between Prometheus Percept and JACK Agent.

Prometheus Percept	JACK Agent
Account Details Request	SalesAssistant
Advanced Search Form Request	SalesAssistant
Basket Plan Request	SalesAssistant
BestSellers Page	SalesAssistant
Book Added to Basket	SalesAssistant
Book Details Page	SalesAssistant
Bookstore Page	SalesAssistant
Categories Request	SalesAssistant
CheckOut Request	SalesAssistant
Contact Information Page	SalesAssistant
Get Delivery Options Page	SalesAssistant
Help Information Page	SalesAssistant
Get Delivery Options Page	SalesAssistant
Help Information Page	SalesAssistant
Log Out Page	SalesAssistant
New Advanced Search	SalesAssistant
New Customer	SalesAssistant
New Keyword Search	SalesAssistant
New Releases Page	SalesAssistant
Place Order Request	SalesAssistant
SignIn Page	SalesAssistant
Special Offers Page	SalesAssistant
Subjects Page	SalesAssistant
Update Basket Request	SalesAssistant

Table D.11 Relations identified manually between Prometheus Percept and JACK Agent

Rule ID	Prometheus Percept	JACK Agent
rulePJ10a	New Advanced Search	SalesAssistant
rulePJ10a	Advanced Search Form Request	SalesAssistant
rulePJ10a	Basket Plan Request	SalesAssistant
rulePJ10a	BestSellers Page	SalesAssistant
rulePJ10a	New Keyword Search	SalesAssistant
rulePJ10a	CheckOut Request	SalesAssistant
rulePJ10a	Account Details Request	SalesAssistant
rulePJ10a	Get Delivery Options Page	SalesAssistant
rulePJ10a	Categories Request	SalesAssistant
rulePJ10a	SignInPage	SalesAssistant
rulePJ10a	Bookstore Page	SalesAssistant
rulePJ10a	New Customer	SalesAssistant
rulePJ10a	Log Out Page	SalesAssistant
rulePJ10a	Book Added to Basket	SalesAssistant
rulePJ10a	Place Order Request	SalesAssistant

Table D.12 Relations identified by the tool between Prometheus Percept and JACK Agent

Rule ID	Prometheus Percept	JACK Agent
RulePJ10cc1	New Advanced Search	
RulePJ10cc1	Advanced Search Form Request	
RulePJ10cc1	Basket Plan Request	
RulePJ10cc1	BestSellers Page	
RulePJ10cc1	New Keyword Search	
RulePJ10cc1	CheckOut Request	
RulePJ10cc1	Account Details Request	
RulePJ10cc1	Get Delivery Options Page	

RulePJ10cc1	Categories Request	
RulePJ10cc1	SignInPage	
RulePJ10cc1	Bookstore Page	
RulePJ10cc1	New Customer	
RulePJ10cc1	Log Out Page	
RulePJ10cc1	Book Added to Basket	
RulePJ10cc1	Place Order Request	
RulePJ10cc1	Book Details Page	
RulePJ10cc1	New Releases Page	
RulePJ10cc1	Special Offers Page	
RulePJ10cc1	Subjects Page	
RulePJ10cc1	Contact Information Page	
RulePJ10cc1	Help Information Page	
RulePJ10cc1	SignIn Page	
RulePJ10cc1	Update Basket Request	

Table D.13 Missing traceability relations between Prometheus Percept and JACK Agent

D.7 JACK Agent vs Prometheus Action

Table D.14 shows the traceability relations identified manually between Prometheus Plan and JACK Agent and table D.15 shows the traceability relations identified by the tool between Prometheus Plan and JACK Agent. The number of relations identified manually is 18, and the number of relations identified by the tool is 10. The precision and recall calculated is 100% and 55.55%, respectively. The reason why recall is low it is because the rule rulePJ11a only capture Actions defined on the System Overview Diagram.

Action	JACK Agent
Show Account Details Page Action	SalesAsssistant
Show Advanced Search Form Page	SalesAsssistant
Show Advanced Search Result Page	SalesAsssistant
Show Basket Page Action	SalesAsssistant
Show BestSellers Page	SalesAsssistant
Show Book Details Page Action	SalesAsssistant
Show Books by Category Page Action	SalesAsssistant
Show Bookstore Home Page	SalesAsssistant
Show Bookstore Page	SalesAsssistant
Show CheckOut Page	SalesAsssistant
Show Contact Information Page	SalesAsssistant
Show Delivery Options Page	SalesAsssistant
Show Help Information Page	SalesAsssistant
Show Keyword Search Result Page	SalesAsssistant
Show New Releases Page	SalesAsssistant
Show Sign In Form Page	SalesAsssistant
Show Special Offers Page Action	SalesAsssistant
Show Subjects Page	SalesAsssistant

Table D.14 Relations identified manually between Prometheus Action and JACK Agent

Rule ID	Prometheus Action	JACK Agent
rulePJ11a	Show Basket Page Action	SalesAssistant
rulePJ11a	Show Advanced Search Form	SalesAssistant

	Page	
rulePJ11a	Show CheckOut Page	SalesAssistant
rulePJ11a	Show Advanced Search Result Page	SalesAssistant
rulePJ11a	Show Keyword Search Result Page	SalesAssistant
rulePJ11a	Show BestSellers Page	SalesAssistant
rulePJ11a	Show Delivery Options Page	SalesAssistant
rulePJ11a	Show Bookstore Home Page	SalesAssistant
rulePJ11a	Show Sign In Form Page	SalesAssistant
rulePJ11a	Show Account Details Page Action	SalesAssistant

Table D.15 Relations identified by the tool between Prometheus Action and JACK Agent

D.8 JACK Agent vs Prometheus Message (sends)

Table D.16 shows the traceability relations identified manually between Prometheus Plan and JACK Agent and table D.17 shows the traceability relations identified by the tool between Prometheus Plan and JACK Agent. The number of relations identified manually is 33, and the number of relations identified by the tool is 4. The precision and recall calculated is 100% and 12.12%, respectively. The reason why recall is low it is because the rule rulePJ12b only capture Message defined on the System Overview Diagram.

Prometheus Message (Sends)	JACK Agent
AdvancedSearch Request	SalesAssistant
Advanced Search Response	StockManager
Authorization Request	SalesAssistant
Authorization Response	PaymentProcessor
BestSellers Request	SalesAssistant
BestSeller Response	StockManager
Book Available	StockManager
Add to Basket Response	StockManager
Book Details Request	SalesAssistant
Book Details Response	StockManager
Add to Basket Request	SalesAssistant
Book Not Available	StockManager
Book Request	SalesAssistant
Books by Category Request	SalesAssistant
Books by Category Response	StockManager
Add Customer Request	SalesAssistant
Delivery Time and Price Response	DeliveryManager
User Details Request	SalesAssistant
Delivery Time and Price Request	SalesAssistant
Keyword Search Request	SalesAssistant
Keyword Search Response	StockManager
New Releases Request	SalesAssistant
New Releases Response	StockManager

Add Customer Response	CustomerRelations
Special Offers Request	SalesAssistant
Special Offers Response	StockManager
Subjects Request	SalesAssistant
Subjects Response	StockManager
Top Ten BestSellers Request	SalesAssistant
Top Ten BestSellers Response	StockManager
User Details Response	CustomerRelations
User Login Request	SalesAssistant
User Login Response	CustomerRelations

Table D.16 Relations identified manually between Prometheus Message and JACK Agent

Rule ID	Prometheus Message	JACK Agent
rulePJ12b	Add Customer Request	SalesAssistant
rulePJ12b	Add to Basket Request	SalesAssistant
rulePJ12b	Add Customer Response	CustomerRelations
rulePJ12b	Add to Basket Response	StockManager

Table D.17 Relations identified by the tool between Prometheus Message and JACK Agent

D.9 JACK Agent vs Prometheus Message (receives)

Table D.18 shows the traceability relations identified manually between Prometheus Plan and JACK Agent and table D.19 shows the traceability relations identified by the tool between Prometheus Plan and JACK Agent. The number of relations identified manually is 34, and the number of relations identified by the tool is 5. The precision and recall calculated is 100% and 14.70%, respectively. The reason why recall is low it is because the rule rulePJ12a only capture Message defined on the System Overview Diagram.

Prometheus Message (Receives)	JACK Agent
AdvancedSearch Request	StockManager
Advanced Search Response	SalesAssistant
Authorization Request	PaymentProcessor
Authorization Response	SalesAssistant
BestSellers Request	StockManager
BestSeller Response	SalesAssistant
Book Avalaible	SalesAssistant
Add to Basket Response	SalesAssistant
Book Details Request	StockManager
Book Details Response	SalesAssistant
Add to Basket Request	StockManager
Book Not Avalaible	SalesAssistant
Book Request	StockManager
Books by Category Request	StockManager
Books by Category Response	SalesAssistant
Add Customer Request	CustomerRelations

Delivery Time and Price Response	SalesAssistant
User Details Request	CustomerRelations
Delivery Time and Price Request	DeliveryManager
Keyword Search Request	StockManager
Keyword Search Response	SalesAssistant
New Releases Request	StockManager
New Releases Response	SalesAssistant
Add Customer Response	SalesAssistant
Special Offers Request	StockManager
Special Offers Response	SalesAssistant
Subjects Request	StockManager
Subjects Response	SalesAssistant
Top Ten BestSellers Request	StockManager
Top Ten BestSellers Response	SalesAssistant
User Details Response	SalesAssistant
User Login Request	CustomerRelations
User Login Response	SalesAssistant
WebSession Request	SalesAssistant

Table D.18 Relations identified manually between Prometheus Message and JACK Agent

Rule ID	Prometheus Message	JACK Agent
rulePJ12a	WebSession Request	SalesAssistant
rulePJ12a	Add Customer Response	SalesAssistant
rulePJ12a	Add to Basket Response	SalesAssistant
rulePJ12a	Add Customer Request	CustomerRelations
rulePJ12a	Add to Basket Request	StockManager

Table D.19 Relations identified by the tool between Prometheus Message and JACK Agent

D.10 JACK Agent vs Prometheus Data (uses)

Table D.20 shows the traceability relations identified manually between Prometheus Data and JACK Agent and table D.21 shows the traceability relations identified by the tool between Prometheus Data and JACK Agent. The number of relations identified manually is 8, and the number of relations identified by the tool is 4. The precision and recall calculated is 100% and 50%, respectively. The reason why recall is low it is because the rule rulePJ13a only capture Percepts defined on the System Overview Diagram.

Prometheus Data (Uses)	JACK Agent
CustomerDB	CustomerRelations
CourierDB	DeliveryManager
BooksDB	StockManager
StockDB	StockManager
BestSellersDB	StockManager

ReleasesDB	StockManager
SpecialOffers	StockManager
Categories	StockManager

Table D.20 Relations identified manually between Prometheus Data and JACK Agent

Rule ID	Prometheus Data	JACK Agent
rulePJ13a	customers	CustomerRelations
rulePJ13a	courier	DeliveryManager
rulePJ13a	books	StockManager
rulePJ13a	categories	StockManager

Table D.21 Relations identified by the tool between Prometheus Data and JACK Agent

D.11 JACK Agent vs Prometheus Data (creates)

Table D.22 shows the traceability relations identified manually between Prometheus Data and JACK Agent and table D.23 shows the traceability relations identified by the tool between Prometheus Data and JACK Agent. The number of relations identified manually is 1, and the number of relations identified by the tool is 1. The precision and recall calculated is 100%.

Prometheus Data (Creates)	JACK Agent
CustomerDB	CustomerRelations

Table D.22 Relations identified manually between Prometheus Data and JACK Agent

Rule ID	Prometheus Data	JACK Agent
rulePJ13b	customers	CustomerRelations

Table D.23 Relations identified by the tool between Prometheus Data and JACK Agent

D.12 JACK Plan vs Prometheus Goal

Table D.24 shows the traceability relations identified manually between JACK Plan and Prometheus Goal and table D.25 shows the traceability relations identified by the tool between JACK Plan and Prometheus Goal. The number of relations identified manually is 39, and the number of relations identified by the tool is 4. The precision and recall calculated is 100% and 10.25%, respectively.

JACK Plan	Prometheus Goal
AddBookToBasket	Retrieve Item Details
RegisterCustomer	Register New Customer
CalculateDeliveryPriceAndTime	Calculate Delivery Time and

	Price
CheckStock	Check Availability
ExecuteAdvancedSearch	Perform Keyword Search
SearchBooksByKeyword	Perform Advanced Search
FindBestSellers	Find BestSellers
FindBookDetails	Find Book Details
FindBooksByCategory	Find Books by Category
FindNewReleases	Find New Releases
FindSpecialOffers	Find Special Offers
FindSubjects	Find Categories
FindTopTenBestSellers	Find Top Ten BestSellers
RespondAddCustomerRequest	Show New Customer Page
RespondAddToBasketRequest	Show Updated Basket
RespondGetDeliveryOptionsRequest	Show Delivery Time and Price Options
RespondLogOutRequest	Show Top Ten BestSellers
RespondPlaceOrderRequest	Show Order Confirmation
RespondSignInRequest	Show Account Details Page
RespondProceedToCheckOutRequest	Proceed To CheckOut
RespondUpdateBasketRequest	Update Basket
GetCreditCardDetails	Get Customer Details
ShowAccount	Show User Details
ShowAdvancedSearchForm	Show Advanced Search Form
ShowAdvancedSearchResult	Show Advanced Search Result
ShowBasket	Show Basket Page
ShowBestSellers	Show Best Sellers Page
ShowBookDetails	Show Book Details Page
ShowBooksByCategory	Show Books by Category Page
ShowWebSite	Show Top Ten BestSellers
ShowContactInfo	Show Contact Information
ShowHelpInfo	Show Help Information
ShowBooksByKeyword	Show Keyword Search Result
ShowNewReleases	Show New Releases
ShowSignInForm	Show LogIn Page
ShowSpecialOffers	Show Special Offers Page
ShowSubjects	Show Categories Page
SignIn	Validate User
ValidateCreditCard	Process Payment

Table D.24 Relations identified manually between JACK Plan and Prometheus Goal

Rule ID	Prometheus Goal	JACK Plan
rulePJ14a	Show Delivery Time and Price Options	RespondGetDeliveryOptionsRequest
rulePJ14a	Show Updated Basket	RespondAddToBasketRequest
rulePJ14a	Perform Advanced Search	ExecuteAdvancedSearch
rulePJ14a	Show Advanced Search Result	ShowAdvancedSearchResult

Table D.25 Relations identified by the tool between Prometheus Goal and JACK Plan

D.13 JACK Plan vs Prometheus Role

Table D.26 shows the traceability relations identified manually between JACK Plan and Prometheus Role and table D.27 shows the traceability relations identified by the tool between Prometheus Plan and JACK Plan. The number of relations identified manually is 39, and the

number of relations identified by the tool is 4. The precision and recall calculated is 100% and 10.25%, respectively.

JACK Plan	Prometheus Role
AddBookToBasket	Search Management
RegisterCustomer	Customer Relationship Management
CalculateDeliveryPriceAndTime	Service Delivery Management
CheckStock	Search Management
ExecuteAdvancedSearch	Stock Management
SearchBooksByKeyword	Stock Management
FindBestSellers	Stock Management
FindBookDetails	Stock Management
FindBooksByCategory	Stock Management
FindNewReleases	Stock Management
FindSpecialOffers	Stock Management
FindSubjects	Stock Management
FindTopTenBestSellers	Stock Management
RespondAddCustomerRequest	Customer Management
RespondAddToBasketRequest	Order Management
RespondGetDeliveryOptionsRequest	Order Management
RespondLogOutRequest	Shop Information Management
RespondPlaceOrderRequest	Order Management
RespondSignInRequest	Customer Management
RespondProceedToCheckOutRequest	Order Management
RespondUpdateBasketRequest	Order Management
GetCreditCardDetails	Customer Relationship Management
ShowAccount	Customer Management
ShowAdvancedSearchForm	Product Information Management
ShowAdvancedSearchResult	Product Information Management
ShowBasket	Order Management
ShowBestSellers	Product Information Management
ShowBookDetails	Product Information Management
ShowBooksByCategory	Product Information Management
ShowWebSite	Shop Information Management
ShowContactInfo	Shop Information Management
ShowHelpInfo	Shop Information Management
ShowBooksByKeyword	Product Information Management
ShowNewReleases	Product Information Management
ShowSignInForm	Customer Management
ShowSpecialOffers	Product Information Management
ShowSubjects	Product Information Management
SignIn	Customer Relationship Management
ValidateCreditCard	Payment Management

Table D.26 Relations identified manually between JACK Plan and Prometheus Role

Rule ID	Prometheus Role	JACK Plan
rulePJ34a	Order Management	RespondGetDeliveryOptionsRequest
rulePJ34a	Order Management	RespondAddToBasketRequest
rulePJ34a	Stock Management	ExecuteAdvancedSearch
rulePJ34a	Product Information Management	ShowAdvancedSearchResult

Table D.27 Relations identified by the tool between Prometheus Role and JACK Plan

D.14 JACK Plan vs Prometheus Agent

Table D.28 shows the traceability relations identified manually between JACK Plan and Prometheus Agent and table D.29 shows the traceability relations identified by the tool

between JACK Plan and Prometheus Agent. The number of relations identified manually is 39, and the number of relations identified by the tool is 34 (the table contains 35 relations, but two of the relations identified by the tool represent the same relation). The precision and recall calculated is 91.17% and 79.49%, respectively.

JACK Plan	Prometheus Agent
AddBookToBasket	Stock Manager
RegisterCustomer	Customer Relations
CalculateDeliveryPriceAndTime	Delivery Manager
CheckStock	Stock Manager
ExecuteAdvancedSearch	Stock Manager
SearchBooksByKeyword	Stock Manager
FindBestSellers	Stock Manager
FindBookDetails	Stock Manager
FindBooksByCategory	Stock Manager
FindNewReleases	Stock Manager
FindSpecialOffers	Stock Manager
FindSubjects	Stock Manager
FindTopTenBestSellers	Stock Manager
RespondAddCustomerRequest	Sales Assistant
RespondAddToBasketRequest	Sales Assistant
RespondGetDeliveryOptionsRequest	Sales Assistant
RespondLogOutRequest	Sales Assistant
RespondPlaceOrderRequest	Sales Assistant
RespondSignInRequest	Sales Assistant
RespondProceedToCheckOutRequest	Sales Assistant
RespondUpdateBasketRequest	Sales Assistant
GetCreditCardDetails	Customer Relations
ShowAccount	Sales Assistant
ShowAdvancedSearchForm	Sales Assistant
ShowAdvancedSearchResult	Sales Assistant
ShowBasket	Sales Assistant
ShowBestSellers	Sales Assistant
ShowBookDetails	Sales Assistant
ShowBooksByCategory	Sales Assistant
ShowWebSite	Sales Assistant
ShowContactInfo	Sales Assistant
ShowHelpInfo	Sales Assistant
ShowBooksByKeyword	Sales Assistant
ShowNewReleases	Sales Assistant
ShowSignInForm	Sales Assistant
ShowSpecialOffers	Sales Assistant
ShowSubjects	Sales Assistant
SignIn	Customer Relations
ValidateCreditCard	Payment Processor
DefaultRequestHandler	Dispatcher Agent
SelectSession	Dispatcher Agent
ShowNewCustomerForm	Sales Assistant
RespondModifyAddressAndPaymentFormRequest	Sales Assistant
Monitor Session	Dispatcher Agent
RespondViewAllOrdersRequest	Sales Assistant

Table D.28 Relations identified manually between JACK Plan and Prometheus Agent

Rule ID	Prometheus Agent	JACK Plan
rulePJ15a	Sales Assistant	RespondAddCustomerRequest
rulePJ15a	Sales Assistant	RespondGetDeliveryOptionsRequest
rulePJ15a	Customer Relations	RespondSignInRequest

rulePJ15a	Customer Relations	ShowSignInForm
rulePJ15a	Customer Relations	SignIn
rulePJ15a	Sales Assistant	RespondSignInRequest
rulePJ15a	Payment Processor	ValidateCreditCard
rulePJ15a	Sales Assistant	RespondPlaceOrderRequest
rulePJ15a	Sales Assistant	RespondViewAllOrdersRequest
rulePJ15a	Sales Assistant	RespondAddToBasketRequest
rulePJ15a	Stock Manager	ExecuteAdvancedSearch
rulePJ15a	Sales Assistant	ShowAdvancedSearchResult
rulePJ15a	Stock Manager	FindBooksByCategory
rulePJ15a	Sales Assistant	ShowBookByCategory
rulePJ15a	Stock Manager	FindBookDetails
rulePJ15a	Sales Assistant	ShowBookDetails
rulePJ15a	Stock Manager	FindBestSellers
rulePJ15a	Stock Manager	FindTopTenBestSellers
rulePJ15a	Sales Assistant	RespondLogoutRequest
rulePJ15a	Stock Manager	FindBestSellers
rulePJ15a	Sales Assistant	ShowBestSellers
rulePJ15a	Stock Manager	FindNewReleases
rulePJ15a	Sales Assistant	ShowNewReleases
rulePJ15a	Stock Manager	FindSpecialOffers
rulePJ15a	Sales Assistant	ShowSpecialOffers
rulePJ15a	Stock Manager	FindSubjects
rulePJ15a	Sales Assistant	ShowSubjects
rulePJ15a	Sales Assistant	RespondProceedToCheckOutRequest
rulePJ15a	Sales Assistant	ShowAccount
rulePJ15a	Sales Assistant	ShowAdvancedSearchForm
rulePJ15a	Sales Assistant	ShowBasket
rulePJ15a	Sales Assistant	ShowContactInfo
rulePJ15a	Sales Assistant	ShowHelpInfo
rulePJ15a	Sales Assistant	ShowSignInForm
rulePJ15a	Sales Assistant	RespondUpdateBasketRequest

Table D.29 Relations identified by the tool between Prometheus Agent and JACK Plan

D.15 JACK Plan vs Prometheus Capability

Table D.30 shows the traceability relations identified manually between JACK Plan and Prometheus Capability and table D.31 shows the traceability relations identified by the tool between JACK Plan and Prometheus Capability. The number of relations identified manually is 39, and the number of relations identified by the tool is 35. The number of corrects relations identified by the tool is 31 and the number of relations missing is 8. The number of relations identified wrong is 4. Precision and recall calculated is 88.57% and 79.49%, respectively.

JACK Plan	Prometheus Capability
AddBookToBasket	Add Item to Basket Capability
RegisterCustomer	Add Customer Capability
CalculateDeliveryPriceAndTime	Calculate Delivery Time and Price Capability
CheckStock	Check Stock Capability
ExecuteAdvancedSearch	Advanced Search Capability
SearchBooksByKeyword	Keyword Search Capability
FindBestSellers	Find BestSellers Capability
FindBookDetails	Find Book Details Capability
FindBooksByCategory	Find Books by Category Capability
FindNewReleases	Find New Releases Capability
FindSpecialOffers	Find Special Offers Capability

FindSubjects	Find Subjects Category Capability
FindTopTenBestSellers	Find Top Ten BestSellers Capability
RespondAddCustomerRequest	Add Customer Response Capability
RespondAddToBasketRequest	Add Item To Basket Response Capability
RespondGetDeliveryOptionsRequest	Get Delivery Options Response Capability
RespondLogOutRequest	Log Out Response Capability
RespondPlaceOrderRequest	Place Order Response Capability
RespondSignInRequest	Login Response Capability
RespondProceedToCheckOutRequest	Proceed To Check Out Response Capability
RespondUpdateBasketRequest	Update Basket Response Capability
GetCreditCardDetails	Retrieve Customer Details Capability
ShowAccount	Show Account Detail Response Capability
ShowAdvancedSearchForm	Show Advanced Search Form Response Capability
ShowAdvancedSearchResult	Advanced Search Response Capability
ShowBasket	Show Basket Response Capability
ShowBestSellers	Show BestSellers Response Capability
ShowBookDetails	Show Book Details Response Capability
ShowBooksByCategory	Show Books by CategoryResponse Caapability
ShowWebSite	Show Bookstore Main Page Response Capability
ShowContactInfo	Show Contact Information Response Capability
ShowHelpInfo	Show Help Information Response Capability
ShowBooksByKeyword	Keyword Search Response Capability
ShowNewReleases	Show New Releases Response Capability
ShowSignInForm	Show SignIn Form Response Capability
ShowSpecialOffers	Show Special Offers Response Capability
ShowSubjects	Show Subjects Response Capability
SignIn	SignIn Capability
ValidateCreditCard	Validate Credit Card Capability

Table D.30 Relations identified manually between JACK Plan and Prometheus Capability

Rule ID	Prometheus Capability	JACK Plan
rulePJ16a	Add Customer Response Capability	RespondAddCustomerRequest
rulePJ16a	Get Delivery Options Response Capability	RespondGetDeliveryOptionsRequest
rulePJ16a	SignIn Capability	RespondSignInRequest
rulePJ16a	SignIn Capability	ShowSignInForm
rulePJ16a	SignIn Capability	SignIn
rulePJ16a	LogIn Response Capability	RespondSignInRequest
rulePJ16a	Validate Credit Card Capability	ValidateCreditCard
rulePJ16a	Place Order Response Capability	RespondPlaceOrderRequest
rulePJ16a	Place Order Response Capability	RespondViewAllOrdersRequest
rulePJ16a	Add Item To Basket Response Capability	RespondAddToBasketRequest
rulePJ16a	Advanced Search Capability	ExecuteAdvancedSearch
rulePJ16a	Advanced Search Response Capability	ShowAdvancedSearchResult
rulePJ16a	Find Books by Category Capabability	FindBooksByCategory
rulePJ16a	Show Books by CategoryResponse Capability	ShowBookByCategory
rulePJ16a	Find Book Details Capability	FindBookDetails
rulePJ16a	Show Book Details Response Capability	ShowBookDetails
rulePJ16a	Find Top Ten BestSellers Capability	FindBestSellers
rulePJ16a	Find Top Ten BestSellers Capability	FindTopTenBestSellers
rulePJ16a	Log Out Response Capability	RespondLogOutRequest
rulePJ16a	Find BestSellers Capability	FindBestSellers
rulePJ16a	Show BestSellers Response Capability	ShowBestSellers
rulePJ16a	Find New Releases Capability	FindNewReleases
rulePJ16a	Show New Releases Response Capability	ShowNewReleases
rulePJ16a	Find Special Offers Capability	FindSpecialOffers
rulePJ16a	Show Special Offers Response Capability	ShowSpecialOffers
rulePJ16a	Find Subjects Category Capability	FindSubjects
rulePJ16a	Show Subjects Response Capability	ShowSubjects
rulePJ16a	Proceed To Check Out Response Capability	RespondProceedToCheckOutRequest
rulePJ16a	Show Account Detail Response Capability	ShowAccount

rulePJ16a	Show Advanced Search Form Response Capability	ShowAdvancedSearchForm
rulePJ16a	Show Basket Response Capability	ShowBasket
rulePJ16a	Show Contact Information Response Capability	ShowContactInfo
rulePJ16a	Show Help Information Response Capability	ShowHelpInfo
rulePJ16a	Show SignIn Form Response Capability	ShowSignInForm
rulePJ16a	Update Basket Response Capability	RespondUpdateBasketRequest

Table D.31 Relations identified by the tool between Prometheus Capability and JACK Plan

D.16 JACK Plan vs Prometheus Plan

Table D.32 shows the traceability relations identified manually between JACK Plan and Prometheus Plan and table D.33 shows the traceability relations identified by the tool between JACK Plan and Prometheus Plan. The number of relations identified manually is 39, and the number of relations identified by the tool is 35. The number of relations identified correctly is 31, the number of relations missing is 8 and the number of relations identified incorrectly by the tool is 4. Precision and recall calculated is 88.57% and 79.49%, respectively.

Prometheus Plan	JACK Plan
Add Item to Basket Plan	AddBookToBasket
Add New Customer	RegisterCustomer
Calculate Delivery Time and Price Plan	CalculateDeliveryPriceAndTime
Check Stock	CheckStock
Execute Advanced Search	ExecuteAdvancedSearch
Execute Keyword Search	SearchBooksByKeyword
Find BestSellers Plan	FindBestSellers
Find Book Details Plan	FindBookDetails
Find Books by Category Plan	FindBooksByCategory
Find New Releases Plan	FindNewReleases
Find Special Offers Plan	FindSpecialOffers
Find Subjects	FindSubjects
Find Top Ten BestSellers Plan	FindTopTenBestSellers
Respond Add Customer Request	RespondAddCustomerRequest
Respond Add Item to Basket Request	RespondAddToBasketRequest
Respond Get Delivery Options Request	RespondGetDeliveryOptionsRequest
Respond Log Out Request	RespondLogOutRequest
Respond Place Order Request	RespondPlaceOrderRequest
Respond SignIn Request	RespondSignInRequest
Respond to Proceed to CheckOut Request	RespondProceedToCheckOutRequest
Respond Update Basket Request	RespondUpdateBasketRequest
Retrieve Customer Details	GetCreditCardDetails
Show Account	ShowAccount
Show Advanced Search Form Plan	ShowAdvancedSearchForm
Show Advanced Search Result Plan	ShowAdvancedSearchResult
Show Basket Plan	ShowBasket
Show BestSellers Plan	ShowBestSellers
Show Book Details	ShowBookDetails
Show Books by Category Plan	ShowBooksByCategory
Show Bookstore Main PagePlan	ShowWebSite
Show Contact Information Plan	ShowContactInfo
Show Help Information Plan	ShowHelpInfo

Show Keyword Search Result Plan	ShowBooksByKeyword
Show New Releases Plan	ShowNewReleases
Show SignIn form	ShowSignInForm
Show Special Offers Plan	ShowSpecialOffers
Show Subjects Plan	ShowSubjects
Sign In	SignIn
Validate Credit Card	ValidateCreditCard
	DefaultRequestHandler
	SelectSession
	ShowNewCustomerForm
	RespondModifyAddressAndPaymentFormRequest
	Monitor Session
	RespondViewAllOrdersRequest

Table D.32 Relations identified manually between Prometheus Plan and JACK Plan

Rule ID	Prometheus Plan	JACK Plan
rulePJ3a	Respond Add Customer Request	RespondAddCustomerRequest
rulePJ3a	Respond Get Delivery Options Request	RespondGetDeliveryOptionsRequest
rulePJ3a	Sign In	RespondSignInRequest
rulePJ3a	Sign In	ShowSignInForm
rulePJ3a	Sign In	SignIn
rulePJ3a	Respond SignIn Request	RespondSignInRequest
rulePJ3a	Validate Credit Card	ValidateCreditCard
rulePJ3a	Respond Place Order Request	RespondPlaceOrderRequest
rulePJ3a	Respond Place Order Request	RespondViewAllOrdersRequest
rulePJ3a	Respond Add Item to Basket Request	RespondAddToBasketRequest
rulePJ3a	Execute Advanced Search	ExecuteAdvancedSearch
rulePJ3a	Show Advanced Search Result Plan	ShowAdvancedSearchResult
rulePJ3a	Find Books by Category Plan	FindBooksByCategory
rulePJ3a	Show Books by Category Plan	ShowBookByCategory
rulePJ3a	Find Book Details Plan	FindBookDetails
rulePJ3a	Show Book Details	ShowBookDetails
rulePJ3a	Find Top Ten BestSellers Plan	FindBestSellers
rulePJ3a	Find Top Ten BestSellers Plan	FindTopTenBestSellers
rulePJ3a	Respond Log Out Request	RespondLogoutRequest
rulePJ3a	Find BestSellers Plan	FindBestSellers
rulePJ3a	Show BestSellers Plan	ShowBestSellers
rulePJ3a	Find New Releases Plan	FindNewReleases
rulePJ3a	Show New Releases Plan	ShowNewReleases
rulePJ3a	Find Special Offers Plan	FindSpecialOffers
rulePJ3a	Show Special Offers Plan	ShowSpecialOffers
rulePJ3a	Find Subjects	FindSubjects
rulePJ3a	Show Subjects Plan	ShowSubjects
rulePJ3a	Respond to Proceed to CheckOut Request	RespondProceedToCheckOutRequest
rulePJ3a	Show Account	ShowAccount
rulePJ3a	Show Advanced Search Form Plan	ShowAdvancedSearchForm
rulePJ3a	Show Basket Plan	ShowBasket
rulePJ3a	Show Contact Information Plan	ShowContactInfo
rulePJ3a	Show Help Information Plan	ShowHelpInfo
rulePJ3a	Show SignIn Form	ShowSignInForm
rulePJ3a	Respond Update Basket Request	RespondUpdateBasketRequest

Table D.33 Relations identified by the tool between Prometheus Plan and JACK Plan

Table D.34 shows a traceability relation that is not identified by the tool. The reason why the tool did not identified the relation is because the element that triggers the Check Stock plan in

Prometheus is Book Request message and the event that triggers the CheckStock plan in JACK is BookRequired and Book Request and BookRequired are not identified as synonymous by the *isTrigger* function.

Prometheus Plan	JACK Plan
Check Stock	CheckStock

Table D.34 Missing relation

Table D.35 shows a traceability relation that is not identified by the tool. The reason why the tool did not identified the relation is because the element that triggers the Calculate Delivery Time and Price Plan plan in Prometheus is Delivery Time and Price Request message and the event that triggers the CalculateDeliveryPriceAndTime plan in JACK is GetDeliveryInformationMessage and Get Delivery Information and Delivery Time and Price Request are not been identified as synonymous by the *isTrigger* function.

Prometheus Plan	JACK Plan
Calculate Delivery Time and Price Plan	CalculateDeliveryPriceAndTime

Table D.35 Missing relation

Table D.36 shows a traceability relation that is not identified by the tool. The element that triggers Sign In plan in Prometheus is WebSession Request message and the event that triggers the RespondSignInRequest plan in JACK is WebSessionRequest. Sign In and RespondSignInRequest are synonyms by the *syn:isSynonyms* function.

The element that triggers Sign In plan in Prometheus is WebSession Request message and the event that triggers the ShowSignInForm plan in JACK is WebSessionRequest. Sign In and RespondSignInRequest are synonyms by the *syn:isSynonyms* function.

rule ID	Prometheus Plan	JACK Plan
rulePJ3a	Sign In	RespondSignInRequest
rulePJ3a	Sign In	ShowSignInForm

Table D.36 Wrong relation

Table D.37 shows a traceability relation that is incorrectly identified by the tool. The element that triggers Find Top Ten BestSellers Plan plan in Prometheus is WebSession Request

message and the event that triggers the FindBestSellers plan in JACK is WebSessionRequest. Find Top Ten BestSellers Plan and FindBestSellers are synonyms by the `syn:isSynonyms` function.

rule ID	Prometheus Plan	JACK Plan
rulePJ3a	Find Top Ten BestSellers Plan	FindBestSellers

Table D.37 Missing relation

Table D.38 shows a traceability relation that is incorrectly identified by the tool. The element that triggers Respond Place Order Request Plan plan in Prometheus is WebSession Request message and the event that triggers the RespondViewAllOrdersRequest plan in JACK is WebSessionRequest. Respond Place Order Request and RespondViewAllOrdersRequest are synonyms by the `syn:isSynonyms` function (Place and Order are synonymous).

rule ID	Prometheus Plan	JACK Plan
rulePJ3a	Respond Place Order Request	RespondViewAllOrdersRequest

Table D.38 Wrong relation

Table D.39 and table D.40 show traceability relations missing between plan in JACK and plan in Prometheus and between plan in Prometheus and plans in JACK, respectively. Table D.41 shows traceability relations that are not identified by the tool, but they were identified manually.

Rule ID	JACK Plan	Prometheus Plan
RulePJ3cc1	AddBookToBasket	
RulePJ3cc1	CalculateDeliveryPriceAndTime	
RulePJ3cc1	CheckStock	
RulePJ3cc1	GetCreditCardDetails	
RulePJ3cc1	MonitorSession	
RulePJ3cc1	RespondModifyAddressAndPaymentFormRequest	
RulePJ3cc1	SearchBooksByKeyword	
RulePJ3cc1	SelectSession	
RulePJ3cc1	ShowBooksByKeyword	
RulePJ3cc1	ShowNewCustomerForm	

Table D.39 Missing relations

Rule ID	Prometheus Plan	JACK Plan
RulePJ3cc1	Add New Customer	
RulePJ3cc1	Retrieve Customer Details	
RulePJ3cc1	Calculate Delivery Time and Price Plan	
RulePJ3cc1		
RulePJ3cc1	Check Stock	
RulePJ3cc1	Add Item to Basket Plan	

RulePJ3cc1	Execute Keyword Search	
RulePJ3cc1	Show Keyword Search Result Plan	
RulePJ3cc1	Show Bookstore Main Page Plan	

Table D.40 Missing relation

Prometheus Plan	JACK Plan
Add Item to Basket Plan	AddBookToBasket
Add New Customer	RegisterCustomer
Calculate Delivery Time and Price Plan	CalculateDeliveryPriceAndTime
Retrieve Customer Details	GetCreditCardDetails
Show Bookstore Main PagePlan	ShowWebSite
Show Keyword Search Result Plan	ShowBooksByKeyword

Table D.41 Relations not identified by the tool

D.17 JACK Plan vs Prometheus Percept

Table D.42 shows the traceability relations identified manually between JACK Plan and Prometheus Percept. Table D.43 shows the traceability relations identified by the tool between JACK Plan and Prometheus Percept. The number of relations identified manually is 23, and the number of relations identified by the tool is 22. The number of relations identified correctly by the tool is 21 and the number of relations missing is 2. The number of relations identified incorrectly by the tool is 1. Precision and recall calculated is 95.45% and 91.30%, respectively.

JACK Plan	Prometheus Percept
RespondAddCustomerRequest	New Customer
RespondAddToBasketRequest	Book Added to Basket
RespondGetDeliveryOptionsRequest	Get Delilvery Options Page
RespondLogOutRequest	Log Out Page
RespondPlaceOrderRequest	Place Order Request
RespondSignInRequest	SignInPage
RespondProceedToCheckOutRequest	CheckOut Request
RespondUpdateBasketRequest	Update Basket Request
GetCreditCardDetails	
ShowAccount	Account Details Request
ShowAdvancedSearchForm	Advanced Search Form Request
ShowAdvancedSearchResult	New Advanced Search
ShowBasket	Basket Plan Request
ShowBestSellers	BestSellers Page
ShowBookDetails	Book Details Page
ShowBooksByCategory	Categories Request
ShowWebSite	Bookstore Page
ShowContactInfo	Contact Information Page
ShowHelpInfo	Help Information Page
ShowBooksByKeyword	New keyword Search
ShowNewReleases	New Releases Page
ShowSignInForm	SignIn Page
ShowSpecialOffers	Special Offers Page
ShowSubjects	Subjects Page

Table D.42 Relations identified manually between JACK Plan and Prometheus Percept

Rule ID	Prometheus Percept	JACK Plan
rulePJ17a	New Customer	RespondAddCustomerRequest
rulePJ17a	Get Delivery Options Page	RespondGetDeliveryOptionsRequest
rulePJ17a	SignInPage	RespondSignInRequest
rulePJ17a	Place Order Request	RespondPlaceOrderRequest
rulePJ17a	Place Order Request	RespondViewAllOrdersRequest
rulePJ17a	Book Added to Basket	RespondAddToBasketRequest
rulePJ17a	New Advanced Search	ShowAdvancedSearchResult
rulePJ17a	Categories Request	ShowBookByCategory
rulePJ17a	Book Details Page	ShowBookDetails
rulePJ17a	Log Out Page	RespondLogOutRequest
rulePJ17a	BestSellers Page	ShowBestSellers
rulePJ17a	New Releases Page	ShowNewReleases
rulePJ17a	Special Offers Page	ShowSpecialOffers
rulePJ17a	Subjects Page	ShowSubjects
rulePJ17a	CheckOut Request	RespondProceedToCheckOutRequest
rulePJ17a	Account Details Request	ShowAccount
rulePJ17a	Advanced Search Form Request	ShowAdvancedSearchForm
rulePJ17a	Basket Plan Request	ShowBasket
rulePJ17a	Contact Information Page	ShowContactInfo
rulePJ17a	Help Information Page	ShowHelpInfo
rulePJ17a	SignIn Page	ShowSignInForm
rulePJ17a	Update Basket Request	RespondUpdateBasketRequest

Table D.43 Relations identified by the tool between Prometheus Percept and JACK Plan

D.18 JACK Plan vs Prometheus Action (Sends)

Table D.44 shows the traceability relations identified manually between JACK Plan and Prometheus Action. Table D.45 shows the traceability relations identified by the tool between JACK Plan and Prometheus Action. The number of relations identified manually is 22, and the number of relations identified by the tool is 21. The number of relations identified correctly is 20 and number of relations missing is 2. The number of relations identified incorrectly by the tool is 1. Precision and recall calculated is 95.23% and 90.00%, respectively.

JACK Plan	Prometheus Action
RespondAddCustomerRequest	Show Account Details Page Action
RespondAddToBasketRequest	Show Account Details Page Action
RespondGetDeliveryOptionsRequest	Show Delivery Options Page
RespondLogOutRequest	Show Bookstore Home Page
RespondPlaceOrderRequest	Show Account Details Page Action
RespondSignInRequest	Show Sign In Form Page
RespondSignInRequest	Show Account Details Page Action
RespondProceedToCheckOutRequest	Show CheckOut Page
RespondUpdateBasketRequest	Show Basket Page Action
ShowAccount	Show Account Details Page Action
ShowAdvancedSearchForm	Show Advanced Search Form Page
ShowAdvancedSearchResult	Show Advanced Search Result Page
ShowBestSellers	Show BestSellers Page
ShowBookDetails	Show Book Details Page Action
ShowBooksByCategory	Show Books by Category Action
ShowWebSite	Show Bookstore Page
ShowContactInfo	Show Contact Information Page

ShowHelpInfo	Show Help Information Page
ShowBooksByKeyword	Show Keyword Search Result Page
ShowNewReleases	Show New Releases Page
ShowSignInForm	Show Sign In Form Page
ShowSpecialOffers	Show Special Offers Page Action
ShowSubjects	Show Subjects Page

Table D.44 Relations identified manually between JACK Plan and Prometheus Action

Rule ID	Prometheus Action	JACK Plan
rulePJ18a	Show Account Details Page Action	RespondAddCustomerRequest
rulePJ18a	Show Delivery Options Page	RespondGetDeliveryOptionsRequest
rulePJ18a	Show Account Details Page Action	RespondSignInRequest
rulePJ18a	Show Sign In Form Page	RespondSignInRequest
rulePJ18a	Show Account Details Page Action	RespondPlaceOrderRequest
rulePJ18a	Show Account Details Page Action	RespondViewAllOrdersRequest
rulePJ18a	Show Advanced Search Result Page	ShowAdvancedSearchResult
rulePJ18a	Show Books by Category Page Action	ShowBookByCategory
rulePJ18a	Show Book Details Page Action	ShowBookDetails
rulePJ18a	Show Bookstore Home Page	RespondLogOutRequest
rulePJ18a	Show BestSellers Page	ShowBestSellers
rulePJ18a	Show New Releases Page	ShowNewReleases
rulePJ18a	Show Special Offers Page Action	ShowSpecialOffers
rulePJ18a	Show Subjects Page	ShowSubjects
rulePJ18a	Show CheckOut Page	RespondProceedToCheckOutRequest
rulePJ18a	Show Account Details Page Action	ShowAccount
rulePJ18a	Show Advanced Search Form Page	ShowAdvancedSearchForm
rulePJ18a	Show Contact Information Page	ShowContactInfo
rulePJ18a	Show Help Information Page	ShowHelpInfo
rulePJ18a	Show Sign In Form Page	ShowSignInForm
rulePJ18a	Show Basket Page Action	RespondUpdateBasketRequest

Table D.45 Relations identified by the tool between Prometheus Action and JACK Plan

D.19 JACK Plan vs Prometheus Message (Sends)

Table D.46 shows the traceability relations identified manually between JACK Plan and Prometheus Message. Table D.47 shows the traceability relations identified by the tool between JACK Plan and Prometheus Message. The number of relations identified manually is 34, and the number of relations identified by the tool is 30. The number of relations identified correctly is 25 and number of relations is 9. The number of relations identified incorrectly by the tool is 5. Precision and recall calculated is 88.33% and 73.52%, respectively.

JACK Plan (Sends)	Prometheus Message
AddBookToBasket	Add to Basket Response
RegisterCustomer	Add Customer Response
CalculateDeliveryPriceAndTime	Delivery Time and Price Response
CheckStock	Book Available
CheckStock	Book Not Available
ExecuteAdvancedSearch	Advanced Search Response
SearchBooksByKeyword	Keyword Search Response
FindBestSellers	BestSellers Response
FindBookDetails	Book Details Response
FindBooksByCategory	Books by Category Response
FindNewReleases	New Releases Response

FindSpecialOffers	Special Offers Response
FindSubjects	Subjects Response
FindTopTenBestSellers	Top Ten BestSellers Response
RespondAddCustomerRequest	Add Customer Request
RespondAddToBasketRequest	Add to Basket Request
RespondGetDeliveryOptionsRequest	User Details Request
RespondGetDeliveryOptionsRequest	Delivery Time and Price Request
RespondLogOutRequest	Top Ten BestSellers Request
RespondPlaceOrderRequest	Authorization Request
RespondPlaceOrderRequest	Book Request
RespondSignInRequest	User Login Request
GetCreditCardDetails	User Details Response
ShowAdvancedSearchResult	Advanced Search Request
ShowBestSellers	BestSellers Request
ShowBookDetails	Book Details Request
ShowBooksByCategory	Books by Category Request
ShowWebSite	Top Ten BestSellers Request
ShowBooksByKeyword	Keyword Search Request
ShowNewReleases	New Releases Request
ShowSpecialOffers	Special Offers Request
ShowSubjects	Subjects Request
SignIn	User Login Response
ValidateCreditCard	Authorization Response

Table D.46 Relations identified manually between JACK Plan and Prometheus Message

Rule ID	Prometheus Message	JACK Plan
rulePJ19a	Add Customer Request	RespondAddCustomerRequest
rulePJ19a	User Details Request	RespondGetDeliveryOptionsRequest
rulePJ19a	Delivery Time and Price Request	RespondGetDeliveryOptionsRequest
rulePJ19a	User Login Response	RespondSignInRequest
rulePJ19a	User Login Response	ShowSignInForm
rulePJ19a	User Login Response	SignIn
rulePJ19a	User Login Request	RespondSignInRequest
rulePJ19a	Authorization Response	ValidateCreditCard
rulePJ19a	Authorization Request	RespondPlaceOrderRequest
rulePJ19a	Book Request	RespondPlaceOrderRequest
rulePJ19a	Authorization Request	RespondViewAllOrdersRequest
rulePJ19a	Book Request	RespondViewAllOrdersRequest
rulePJ19a	Add to Basket Request	RespondAddToBasketRequest
rulePJ19a	Advanced Search Response	ExecuteAdvancedSearch
rulePJ19a	Advanced Search Request	ShowAdvancedSearchResult
rulePJ19a	Books by Category Response	FindBooksByCategory
rulePJ19a	Books by Category Request	ShowBookByCategory
rulePJ19a	Book Details Response	FindBookDetails
rulePJ19a	Book Details Request	ShowBookDetails
rulePJ19a	Top Ten BestSellers Response	FindBestSellers
rulePJ19a	Top Ten BestSellers Response	FindTopTenBestSellers
rulePJ19a	Top Ten BestSellers Request	RespondLogOutRequest
rulePJ19a	BestSellers Response	FindBestSellers
rulePJ19a	BestSellers Request	ShowBestSellers
rulePJ19a	New Releases Response	FindNewReleases
rulePJ19a	New Releases Request	ShowNewReleases
rulePJ19a	Special Offers Response	FindSpecialOffers
rulePJ19a	Special Offers Request	ShowSpecialOffers
rulePJ19a	Subjects Response	FindSubjects
rulePJ19a	Subjects Request	ShowSubjects

Table D.47 Relations identified by the tool between Prometheus Message and JACK Plan

D.20 JACK Plan vs Prometheus Message (Receives)

Table D.48 shows the traceability relations identified manually between JACK Plan and Prometheus Message. Table D.49 shows the traceability relations identified by the tool between JACK Plan and Prometheus Message. The number of relations identified manually is 34, and the number of relations identified by the tool is 30. The number of relations identified correctly is 25 and number of relations missing is 9. The number of relations identified incorrectly by the tool is 5. Precision and recall calculated is 88.33% and 73.52%, respectively.

JACK Plan (Receives)	Prometheus Message
AddBookToBasket	Add to Basket Request
RegisterCustomer	Add Customer Request
CalculateDeliveryPriceAndTime	Delivery Time and Price Request
CheckStock	Book Request
ExecuteAdvancedSearch	Advanced Search Request
SearchBooksByKeyword	Keyword Search Request
FindBestSellers	BestSellers Request
FindBookDetails	Book Details Request
FindBooksByCategory	Books by Category Request
FindNewReleases	New Releases Request
FindSpecialOffers	Special Offers Request
FindSubjects	Subjects Request
FindTopTenBestSellers	Top Ten BesSellers Request
RespondAddCustomerRequest	WebSession Request
RespondAddCustomerRequest	Add Customer Response
RespondAddToBasketRequest	WebSession Request
RespondAddToBasketRequest	Add to Basket Response
RespondGetDeliveryOptionsRequest	WebSession Request
RespondGetDeliveryOptionsRequest	Delivery Time and Price Response
RespondGetDeliveryOptionsRequest	User Details Response
RespondLogOutRequest	WebSession Request
RespondLogOutRequest	Top Ten BestSellers Response
RespondPlaceOrderRequest	WebSession Request
RespondPlaceOrderRequest	Authorization Response
RespondPlaceOrderRequest	Book Available
RespondPlaceOrderRequest	Book Not Available
RespondSignInRequest	WebSession Request
RespondSignInRequest	User Login Response
RespondProceedToCheckOutRequest	WebSession Request
RespondUpdateBasketRequest	WebSession Request
GetCreditCardDetails	User Details Request
ShowAccount	WebSession Request
ShowAdvancedSearchForm	WebSession Request
ShowAdvancedSearchResult	WebSession Request
ShowAdvancedSearchResult	Advanced Search Response
ShowBasket	WebSession Request
ShowBestSellers	WebSession Request
ShowBestSellers	BestSellers Response
ShowBookDetails	WebSession Request
ShowBookDetails	Book Details Response
ShowBooksByCategory	WebSession Request
ShowBooksByCategory	Books by Category Response
ShowWebSite	WebSession Request
ShowWebSite	Top Ten BestSellers Response
ShowContactInfo	WebSession Request
ShowHelpInfo	WebSession Request
ShowBooksByKeyword	WebSession Request
ShowBooksByKeyword	Keyword Search Response
ShowNewReleases	WebSession Request
ShowNewReleases	New Releases Response
ShowSignInForm	WebSession Request

ShowSpecialOffers	WebSession Request
ShowSpecialOffers	Special Offers Response
ShowSubjects	WebSession Request
ShowSubjects	Subjects Response
SignIn	User Login Request
ValidateCreditCard	Authorization Request

Table D.48 Relations identified manually between JACK Plan and Prometheus Message

Prometheus Message	JACK Plan
Add Customer Response	RespondAddCustomerRequest
User Details Response	RespondGetDeliveryOptionsRequest
Delivery Time and Price Response	RespondGetDeliveryOptionsRequest
User Login Response	RespondSignInRequest
Authorization Response	RespondPlaceOrderRequest
Book Available	RespondPlaceOrderRequest
Book Not Available	RespondPlaceOrderRequest
Authorization Response	RespondViewAllOrdersRequest
Book Available	RespondViewAllOrdersRequest
Book Not Available	RespondViewAllOrdersRequest
Add to Basket Response	RespondAddToBasketRequest
Advanced Search Response	ShowAdvancedSearchResult
Books by Category Response	ShowBookByCategory
Book Details Response	ShowBookDetails
Top Ten BestSellers Response	RespondLogoutRequest
BestSellers Response	ShowBestSellers
New Releases Response	ShowNewReleases
Special Offers Response	ShowSpecialOffers
Subjects Response	ShowSubjects
WebSession Request	RespondAddCustomerRequest
WebSession Request	RespondGetDeliveryOptionsRequest
User Login Request	RespondSignInRequest
User Login Request	ShowSignInForm
User Login Request	SignIn
WebSession Request	RespondSignInRequest
Authorization Request	ValidateCreditCard
WebSession Request	RespondPlaceOrderRequest
WebSession Request	RespondViewAllOrdersRequest
WebSession Request	RespondAddToBasketRequest
Advanced Search Request	ExecuteAdvancedSearch
WebSession Request	ShowAdvancedSearchResult
Books by Category Request	FindBooksByCategory
WebSession Request	ShowBookByCategory
Book Details Request	FindBookDetails
WebSession Request	ShowBookDetails
Top Ten BestSellers Request	FindBestSellers
Top Ten BestSellers Request	FindTopTenBestSellers
WebSession Request	RespondLogoutRequest
BestSellers Request	FindBestSellers
WebSession Request	ShowBestSellers
New Releases Request	FindNewReleases
WebSession Request	ShowNewReleases
Special Offers Request	FindSpecialOffers
WebSession Request	ShowSpecialOffers
Subjects Request	FindSubjects
WebSession Request	ShowSubjects
WebSession Request	RespondProceedToCheckOutRequest
WebSession Request	ShowAccount
WebSession Request	ShowAdvancedSearchForm
WebSession Request	ShowBasket
WebSession Request	ShowContactInfo
WebSession Request	ShowHelpInfo
WebSession Request	ShowSignInForm
WebSession Request	RespondUpdateBasketRequest

Table D.49 Relations identified by the tool between Prometheus Message and JACK Plan

D.21 JACK Plan vs Prometheus Data (Uses)

Table D.51 shows the traceability relations identified manually between JACK Plan and Prometheus Data. Table D.52 shows the traceability relations identified by the tool between JACK Plan and Prometheus Data. The number of relations identified manually is 19, and the number of relations identified by the tool is 12. The number of relations identified correctly is 10 and number of relations missing is 9. The number of relations identified incorrectly by tool is 2. Precision and recall calculated is 83.33% and 52.63%, respectively.

JACK Plan (Uses)	Prometheus Data
AddBookToBasket	BooksDB
CalculateDeliveryPriceAndTime	CourierDB
CheckStock	BooksDB
CheckStock	StockDB
ExecuteAdvancedSearch	BooksDB
ExecuteAdvancedSearch	StockDB
SearchBooksByKeyword	BooksDB
SearchBooksByKeyword	StockDB
FindBestSellers	BestSellersDB
FindBookDetails	StockDB
FindBookDetails	BooksDB
FindBooksByCategory	BooksDB
FindBooksByCategory	StockDB
FindNewReleases	NewReleasesDB
FindSpecialOffers	SpecialOffersDB
FindSubjects	CategoriesDB
FindTopTenBestSellers	BestSellersDB
GetCreditCardDetails	CustomerDB
SignIn	CustomerDB

Table D.50 Relations identified manually between JACK Plan and Prometheus Data

Rule ID	Prometheus Data	JACK Plan
rulePJ20a	customers	RespondSignInRequest
rulePJ20a	customers	ShowSignInForm
rulePJ20a	customers	SignIn
rulePJ20a	books	ExecuteAdvancedSearch
rulePJ20a	stock	ExecuteAdvancedSearch
rulePJ20a	books	FindBooksByCategory
rulePJ20a	stock	FindBookDetails
rulePJ20a	bestsellers	FindBestSellers
rulePJ20a	bestsellers	FindTopTenBestSellers
rulePJ20a	bestsellers	FindBestSellers
rulePJ20a	new releases	FindNewReleases
rulePJ20a	special offers	FindSpecialOffers
rulePJ20a	categories	FindSubjects

Table D.51 Relations identified by tool between JACK Plan and Prometheus Data

D.22 JACK Plan vs Prometheus Data (Creates)

Table D.52 shows the traceability relations identified manually between JACK Plan and Prometheus Data. The number of relations identified manually is 1, and the number of relations identified by the tool is 0. Therefore, the precision and recall calculated is 100.0% and 0%, respectively.

JACK Plan (Creates)	Prometheus Data
RegisterCustomer	CustomerDB

Table D.52 Relations identified manually between JACK Plan and Prometheus Data

D.23 JACK BeliefSet vs Prometheus Role (Creates)

Table D.53 shows traceability relations identified manually between JACK BeliefSet and Prometheus Role. Table D.54 shows traceability relations identified by the tool between Prometheus Role and JACK BeliefSet. The number of relations identified manually is 1, and the number of relations identified by the tool is 2. Therefore, precision and recall calculated is 50.0% and 100%, respectively.

JACK BeliefSet	Prometheus Role
CustomerDB	Customer Relationship Management

Table D.53 Relations identified manually between JACK BeliefSet and Prometheus Role

Rule ID	Prometheus Role	JACK BeliefSet
rulePJ21b	Customer Relationship Management	CustomerDB
rulePJ21b	Customer Relationship Management	CustomerOrderDB

Table D.54 Relations identified by the tool between Prometheus Role and JACK BeliefSet

D.24 JACK BeliefSet vs Prometheus Role (Uses)

Table D.55 shows traceability relations identified manually between Prometheus Role and JACK BeliefSet. Table D.56 shows traceability relations identified by the tool between Prometheus Role and JACK BeliefSet. The number of relations identified manually is 10, and the number of relations identified by the tool is 11. Therefore, precision and recall calculated is 90.90% and 100%, respectively.

Prometheus Role	JACK BeliefSet
Stock Management	BooksDB
Stock Management	StockDB
Stock Management	BestSellersDB
Stock Management	NewReleasesDB
Stock Management	SpecialOffersDB
Stock Management	CategoriesDB
Search Management	BooksDB
Search Management	StockDB
Service Delivery Management	CourierDB
Customer Relations Management	CustomerDB

Table D.55 Relations identified manually between JACK BeliefSet and Prometheus Role

Rule ID	Prometheus Role	JACK BeliefSet
rulePJ21a	Stock Management	BooksDB
rulePJ21a	Search Management	BooksDB
rulePJ21a	Customer Relationship Management	CustomerDB
rulePJ21a	Customer Relationship Management	CustomerOrderDB
rulePJ21a	Service Delivery Management	CourierDB
rulePJ21a	Stock Management	StockDB
rulePJ21a	Search Management	StockDB
rulePJ21a	Stock Management	CategoriesDB
rulePJ21a	Stock Management	BestSellersDB
rulePJ21a	Stock Management	NewReleasesDB
rulePJ21a	Stock Management	SpecialOffersDB

Table D.56 – Relations identified by the tool between JACK BeliefSet and Prometheus Role

D.25 JACK BeliefSet vs Prometheus Role (Creates)

Table D.57 shows traceability relations identified manually between JACK BeliefSet and Prometheus Agent. Table D.58 shows traceability relations identified by the tool between Prometheus Role and JACK BeliefSet. The number of relations identified manually is 1, and the number of relations identified by the tool is 2. Therefore, precision and recall calculated is 50% and 100%, respectively.

JACK BeliefSet	Prometheus Role
CustomerDB	Customer Relationship Management

Table D.57 Relations identified manually between JACK BeliefSet and Prometheus Agent

Rule ID	Prometheus Role	JACK BeliefSet
rulePJ21b	Customer Relationship Management	CustomerDB
rulePJ21b	Customer Relationship Management	CustomerOrderDB

Table D.58 Relations identified by the tool between Prometheus Role

D.26 JACK BeliefSet vs Prometheus Agent (Uses)

Table D.59 shows traceability relations identified manually between JACK BeliefSet and Prometheus Agent. Table D.60 shows traceability relations identified by the tool between Prometheus Agent and JACK BeliefSet. The number of relations identified manually is 8, and the number of relations identified by the tool is 5. Therefore, precision and recall calculated is 80% and 50%, respectively.

JACK BeliefSet	Prometheus Agent
BestSellersDB	Stock Manager
BooksDB	Stock Manager
CategoriesDB	Stock Manager
CourierDB	Delivery Manager
CustomerDB	Customer Relations
NewReleasesDB	Stock Manager
SpecialOffersDB	Stock Manager
StockDB	Stock Manager
CustomerOrderDB	
Session.bel	

Table D.59 Relations identified manually between JACK BeliefSet and Prometheus Agent

Rule ID	Prometheus Agent	JACK BeliefSet
rulePJ22a	Customer Relations	CustomerDB
rulePJ22a	Customer Relations	CustomerOrderDB
rulePJ22a	Delivery Manager	CourierDB
rulePJ22a	Stock Manager	BooksDB
rulePJ22a	Stock Manager	CategoriesDB

Table D.60 Relations identified by the tool between Prometheus Agent and JACK BeliefSet

D.27 JACK BeliefSet vs Prometheus Capability (Creates)

Table D.61 shows traceability relations identified manually between Prometheus Capability and JACK BeliefSet. Table D.62 shows traceability relations identified by the tool between Prometheus Capability and JACK BeliefSet. The number of relations identified manually is 1, and the number of relations identified by the tool is 2. Therefore, precision and recall calculated is 50% and 100%, respectively.

Prometheus Capability	JACK BeliefSet
Add Customer Capability	CustomerDB

Table D.61 Relations identified manually between Prometheus Capability and JACK BeliefSet

Rule ID	Prometheus Capability	JACK BeliefSet
rulePJ23b	Add Customer Capability	CustomerDB
rulePJ23b	Add Customer Capability	CustomerOrderDB

Table D.62 Relations identified by the tool between Prometheus Capability and JACK BeliefSet

D.28. JACK BeliefSet vs Prometheus Capability (Uses)

Table D.63 shows traceability relations identified manually between Prometheus Capability and JACK BeliefSet. Table D.64 shows traceability relations identified by the tool between Prometheus Capability and JACK BeliefSet. The number of relations identified manually is 19, and the number of relations identified by the tool is 21. The number of relations identified correctly is 19, the number of relations missing is 0, and the number of relations identified incorrectly is 2. Precision and recall calculated is 90.47% and 100%, respectively.

Prometheus Capability	JACK BeliefSet
Add Item to Basket Capability	BookDB
Retrieve Customer Details Capability	CustomerDB
SignIn Capability	CustomerDB
Calculate Delivery Time and Price Capability	CourierDB
Advanced Search Capability	BookDB
Advanced Search Capability	StockDB
Keyword Search Capability	StockDB
Keyword Search Capability	BookDB
Find Top Ten BestSellers Capability	BestSellersDB
Check Stock Capability	StockDB
Check Stock Capability	BookDB
Find BestSellers Capability	BestSellersDB
Find Books by Category Capability	BookDB
Find Books by Category Capability	StockDB
Find Book Details Capability	StockDB
Find Book Details Capability	BookDB
Find New Releases Capability	NewReleasesDB
Find Special Offers Capability	SpecialOffersDB
Find Subjects Category Capability	CategoriesDB

Table D.63 Relations identified manually between Prometheus Capability and JACK BeliefSet

Rule ID	Prometheus Capability	JACK BeliefSet
rulePJ23a	Add Item to Basket Capability	BooksDB
rulePJ23a	Check Stock Capability	BooksDB
rulePJ23a	Advanced Search Capability	BooksDB
rulePJ23a	Keyword Search Capability	BooksDB
rulePJ23a	Find Books by Category Capabability	BooksDB
rulePJ23a	Find Book Details Capability	BooksDB
rulePJ23a	Retrieve Customer Details Capability	CustomerDB
rulePJ23a	SignIn Capability	CustomerDB
rulePJ23a	Retrieve Customer Details Capability	CustomerOrderDB

rulePJ23a	SignIn Capability	CustomerOrderDB
rulePJ23a	Calculate Delivery Time and Price Capability	CourierDB
rulePJ23a	Check Stock Capability	StockDB
rulePJ23a	Advanced Search Capability	StockDB
rulePJ23a	Keyword Search Capability	StockDB
rulePJ23a	Find Books by Category Capabability	StockDB
rulePJ23a	Find Book Details Capability	StockDB
rulePJ23a	Find Subjects Category Capability	CategoriesDB
rulePJ23a	Find Top Ten BestSellers Capability	BestSellersDB
rulePJ23a	Find BestSellers Capability	BestSellersDB
rulePJ23a	Find New Releases Capability	NewReleasesDB
rulePJ23a	Find Special Offers Capability	SpecialOffersDB

Table D.64 Relations identified by the tool between Prometheus Capability and JACK BeliefSet

D.29 JACK BeliefSet vs Prometheus Plan (Creates)

Table D.65 shows traceability relations identified manually between Prometheus Plan and JACK BeliefSet. Table D.66 shows traceability relations identified by the tool between Prometheus Plan and JACK BeliefSet. The number of relations identified manually is 1, and the number of relations identified by the tool is 2. The number of relations identified correctly is 1, the number of relations missing is 0, and the number of relations identified incorrectly is 1. Precision and recall calculated is 50% and 100%, respectively.

Prometheus Plan	JACK BeliefSet
Add New Customer	CustomerDB

Table D.65 Relations identified manually between Prometheus Plan and JACK

Rule ID	Prometheus Plan	JACK BeliefSet
rulePJ24b	Add New Customer	CustomerDB
rulePJ24b	Add New Customer	CustomerOrderDB

Table D.66 Relations identified by the tool between Prometheus Plan and JACK BeliefSet

D.30 JACK BeliefSet vs Prometheus Plan (Uses)

Table D.67 shows traceability relations identified manually between Prometheus Plan and JACK BeliefSet. Table D.68 shows traceability relations identified by the tool between Prometheus Plan and JACK BeliefSet. The number of relations identified manually is 19, and the number of relations identified by the tool is 21. The number of relations identified correctly is 19, the number of relations missing is 0, and the number of relations identified incorrectly is 2. Precision and recall calculated is 90.47% and 100%, respectively.

Prometheus Plan	JACK BeliefSet
Add Item to Basket Plan	BookDB
Calculate Delivery Time and Price Plan	CourierDB
Check Stock	StockDB
Check Stock	BookDB
Execute Advanced Search	BooksDB
Execute Advanced Search	StockDB
Execute Keyword Search	BooksDB
Execute Keyword Search	StockDB
Find BestSellers Plan	BestSellersDB
Find Book Details Plan	StockDB
Find Book Details Plan	BookDB
Find Books by Category Plan	BookDB
Find Books by Category Plan	StockDB
Find New Releases Plan	NewReleasesDB
Find Special Offers Plan	SpecialOffersDB
Find Subjects	CategoriesDB
Find Top Ten BestSellers Plan	BestSellersDB
Retrieve Customer Details	CustomerDB
Sign In	CustomerDB

Table D.67 Relations identified manually between Prometheus Plan and JACK BeliefSet

Rule ID	Prometheus Plan	JACK BeliefSet
rulePJ24a	Retrieve Customer Details	CustomerDB
rulePJ24a	Sign In	CustomerDB
rulePJ24a	Retrieve Customer Details	CustomerOrderDB
rulePJ24a	Sign In	CustomerOrderDB
rulePJ24a	Calculate Delivery Time and Price Plan	CourierDB
rulePJ24a	Check Stock	BooksDB
rulePJ24a	Add Item to Basket Plan	BooksDB
rulePJ24a	Execute Advanced Search	BooksDB
rulePJ24a	Execute Keyword Search	BooksDB
rulePJ24a	Find Books by Category Plan	BooksDB
rulePJ24a	Find Book Details Plan	BooksDB
rulePJ24a	Check Stock	StockDB
rulePJ24a	Execute Advanced Search	StockDB
rulePJ24a	Execute Keyword Search	StockDB
rulePJ24a	Find Books by Category Plan	StockDB
rulePJ24a	Find Book Details Plan	StockDB
rulePJ24a	Find Top Ten BestSellers Plan	BestSellersDB
rulePJ24a	Find BestSellers Plan	BestSellersDB
rulePJ24a	Find New Releases Plan	NewReleasesDB
rulePJ24a	Find Special Offers Plan	SpecialOffersDB
rulePJ24a	Find Subjects	CategoriesDB

Table D.68 Relations identified by the tool between Prometheus Plan and JACK BeliefSet

D.31 JACK BeliefSet vs Prometheus Data

Table D.69 shows traceability relations identified manually between Prometheus Data and JACK BeliefSet. Table D.70 shows traceability relations identified by the tool between Prometheus Data and JACK BeliefSet. The number of relations identified manually is 8, and the number of relations identified by the tool is 9. The number of relations identified correctly is 8, the number of relations missing is 0, and the number of relations identified incorrectly is 1. Precision and recall calculated is 88.88% and 100%, respectively.

Prometheus Data	JACK BeliefSet
BestSellersDB	BestSellersDB
BookDB	BooksDB
CategoriesDB	CategoriesDB
CourierDB	CourierDB
CustomerDB	CustomerDB
NewReleasesDB	NewReleasesDB
SpecialOffersDB	SpecialOffersDB
StockDB	StockDB
	CustomerOrderDB
	Session.bel

Table D.69 Relations between JACK BeliefSet and Prometheus Data

Rule ID	Prometheus Data	JACK BeliefSet
rulePJ2a	BookDB	BooksDB
rulePJ2a	StockDB	StockDB
rulePJ2a	BestSellersDB	BestSellersDB
rulePJ2a	NewReleasesDB	NewReleasesDB
rulePJ2a	SpecialOffersDB	SpecialOffersDB
rulePJ2a	CategoriesDB	CategoriesDB
rulePJ2a	CustomerDB	CustomerDB
rulePJ2a	CustomerDB	CustomerOrderDB
rulePJ2a	CourierDB	CourierDB

Table D.70 Relations between Prometheus Data and JACK BeliefSet

D.32 JACK Event vs Prometheus Agent (sends)

Table D.71 shows traceability relations identified manually between JACK Event and Prometheus Agent. The number of relations identified manually is 33, and the number of relations identified by the tool is 0.

JACK Event	Prometheus Agent
AdvancedSearchRequest	SalesAssistant
AdvancedSearchResponse	StockManager
AuthorizationRequest	SalesAssistant
AuthorizationResponse	PaymentProcessor
BestSellersRequest	SalesAssistant
BestSellersResponse	StockManager
BookAvalaible	StockManager
BookDetails	StockManager
BookDetailsRequest	SalesAssistant
BookDetailsResponse	StockManager
BookInBasket	SalesAssistant
BookNotAvalaible	StockManager
BookRequired	SalesAssistant
BooksByCategoryRequest	SalesAssistant
BooksByCategoryResponse	StockManager
CustomerDetails	SalesAssistant
DeliveryOptionsInformation	DeliveryManager
DeliveryOptionsRequest	SalesAssistant
GetDeliveryInformationMessage	SalesAssistant
KeywordSearchRequest	SalesAssistant
KeywordSearchResponse	StockManager
NewReleasesRequest	SalesAssistant
NewReleasesResponse	StockManager
RegisterCustomerResponse	CustomerRelations
SessionAccess	

SpecialOffersRequest	SalesAssistant
SpecialOffersResponse	StockManager
SubjectsRequest	SalesAssistant
SubjectsResponse	StockManager
TopTenBestSellersRequest	SalesAssistant
TopTenBestSellersResponse	StockManager
UserDetails	CustomerRelations
UserLoginRequest	SalesAssistant
UserLoginResponse	CustomerRelations
WebDispatch	

Table D.71 Relations between JACK Event and Prometheus Agent

D.33 JACK Event vs Prometheus Agent (receives)

Table D.72 shows traceability relations identified manually between JACK Event and Prometheus Agent. The number of relations identified manually is 33, and the number of relations identified by the tool is 0.

JACK Event	Prometheus Agent
AdvancedSearchRequest	StockManager
AdvancedSearchResponse	SalesAssistant
AuthorizationRequest	PaymentProcessor
AuthorizationResponse	SalesAssistant
BestSellersRequest	StockManager
BestSellersResponse	SalesAssistant
BookAvailable	SalesAssistant
BookDetails	SalesAssistant
BookDetailsRequest	StockManager
BookDetailsResponse	SalesAssistant
BookInBasket	StockManager
BookNotAvailable	SalesAssistant
BookRequired	StockManager
BooksByCategoryRequest	StockManager
BooksByCategoryResponse	SalesAssistant
CustomerDetails	CustomerRelations
DeliveryOptionsInformation	SalesAssistant
DeliveryOptionsRequest	CustomerRelations
GetDeliveryInformationMessage	DeliveryManager
KeywordSearchRequest	StockManager
KeywordSearchResponse	SalesAssistant
NewReleasesRequest	StockManager
NewReleasesResponse	SalesAssistant
RegisterCustomerResponse	SalesAssistant
SessionAccess	DispatcherAgent
SpecialOffersRequest	StockManager
SpecialOffersResponse	SalesAssistant
SubjectsRequest	StockManager
SubjectsResponse	SalesAssistant
TopTenBestSellersRequest	StockManager
TopTenBestSellersResponse	SalesAssistant
UserDetails	SalesAssistant
UserLoginRequest	CustomerRelations
UserLoginResponse	SalesAssistant
WebDispatch	DispatcherAgent
	SalesAssistant
RegisterCustomerResponse	

Table D.72 Relations between JACK Event and Prometheus Agent

D.34 JACK Event vs Prometheus Capability (sends)

Table D.73 shows traceability relations identified manually between JACK Event and Prometheus Agent. Table D.74 shows traceability relations identified by the tool between JACK Event and Prometheus Capability. The number of relations identified manually is 33, and the number of relations identified by the tool is 0.

JACK Event	Prometheus Capability
AdvancedSearchRequest	Advanced Search Response Capability
AdvancedSearchResponse	Advanced Search Capability
AuthorizationRequest	Place Order Response Capability
AuthorizationResponse	Validate Credit Card Capability
BestSellersRequest	Show BestSellers Response Capability
BestSellersResponse	Find BestSellers Capability
BookAvalaible	Check Stock Capability
BookDetails	Add Item to Basket Capability
BookDetailsRequest	Show Book Details Response Capability
BookDetailsResponse	Find Book Details Capability
BookInBasket	Add Item to Basket Response Capability
BookNotAvalaible	Check Stock Capability
BookRequired	Place Order Response Capability
BooksByCategoryRequest	Show Books by CategoryResponse Capability
BooksByCategoryResponse	Find Books by Category Capability
CustomerDetails	Add Customer Response Capability
DeliveryOptionsInformation	Calculate Delivery Time and Price Capability
DeliveryOptionsRequest	Get Delivery Options Response Capability
GetDeliveryInformationMessage	Get Delivery Options Response Capability
KeywordSearchRequest	Keyword Search Response Capability
KeywordSearchResponse	
NewReleasesRequest	Show New Releases Response Capability
NewReleasesResponse	Find New Releases Capability
RegisterCustomerResponse	Add Customer Capability
SessionAccess	
SpecialOffersRequest	Show Special Offers Response Capability
SpecialOffersResponse	Find Special Offers Capability
SubjectsRequest	Show Subjects Response Capability
SubjectsResponse	Find Subjects Category Capability
TopTenBestSellersRequest	Show Bookstore Main Page Response Capability
TopTenBestSellersResponse	Find Top Ten BestSellers Capability
UserDetails	Retrieve Customer Details Capability
UserLoginRequest	Login Response Capability
UserLoginResponse	SignIn Capability
WebDispatch	
RegisterCustomerResponse	

Table D.73 Relations between JACK Event and Prometheus Capability

D.35. JACK Event vs Prometheus Capability (receives)

Table D.74 shows traceability relations identified manually between JACK Event and Prometheus Capability. Table D.74 shows traceability relations identified by the tool between

JACK Event and Prometheus Capability. The number of relations identified manually is 33, and the number of relations identified by the tool is 0.

JACK Event	Prometheus Capability
AdvancedSearchRequest	Advanced Search Capability
AdvancedSearchResponse	Advanced Search Response Capability
AuthorizationRequest	Validate Credit Card Capability
AuthorizationResponse	Place Order Response Capability
BestSellersRequest	Find BestSellers Capability
BestSellersResponse	Show BestSellers Response Capability
BookAvalaible	Place Order Response Capability
BookDetails	Add Item to Basket Capability
BookDetailsRequest	Find Book Details Capability
BookDetailsResponse	Show Book Details Response Capability
BookInBasket	Add Item to Basket Capability
BookNotAvalaible	Place Order Response Capability
BookRequired	Check Stock Capability
BooksByCategoryRequest	Find Books by Category Capability
BooksByCategoryResponse	Show Books by CategoryResponse Capability
CustomerDetails	Add Customer Capability
DeliveryOptionsInformation	Get Delivery Options Response Capability
DeliveryOptionsRequest	Retrieve Customer Details
GetDeliveryInformationMessage	Calculate Delivery Time and Price Capability
KeywordSearchRequest	Keyword Search Capability
KeywordSearchResponse	Keyword Search Response Capability
NewReleasesRequest	Find New Releases Capability
NewReleasesResponse	Show New Releases Response Capability
RegisterCustomerResponse	Add Customer Response Capability
SessionAccess	
SpecialOffersRequest	Find Special Offers Capability
SpecialOffersResponse	Show Special Offers Response Capability
SubjectsRequest	Find Subjects Category Capability
SubjectsResponse	Show Subjects Response Capability
TopTenBestSellersRequest	Find Top Ten BestSellers Capability
TopTenBestSellersResponse	Show Bookstore Main Page Response Capability
UserDetails	Get Delivery Options Response Capability
UserLoginRequest	SignIn Capability
UserLoginResponse	LogIn Response Capability

Table D.74 Relations identified manually between JACK Event and Prometheus Capability

D.36 JACK Event vs Prometheus Plan (sends)

Table D.75 shows traceability relations identified manually between JACK Event and Prometheus Plan. Table D.76 shows traceability relations identified by the tool between JACK Event and Prometheus Plan. The number of relations identified manually is 33, and the

number of relations identified by the tool is 0. Precision and recall calculated is 100% and 55.88%, respectively.

JACK Event	Prometheus Plan
AdvancedSearchRequest	Show Advanced Search Result Plan
AdvancedSearchResponse	Execute Advanced Search
AuthorizationRequest	Respond Place Order Request
AuthorizationResponse	Validate Credit Card
BestSellersRequest	Show BestSellers Plan
BestSellersResponse	Find BestSellers Plan
BookAvalaible	Check Stock
BookDetails	Add Item to Basket Plan
BookDetailsRequest	Show Book Details
BookDetailsResponse	Find Book Details Plan
BookInBasket	Respond Add Item to Basket Request
BookNotAvalaible	Check Stock
BookRequired	Respond Place Order Request
BooksByCategoryRequest	Show Books by Category Plan
BooksByCategoryResponse	Find Books by Category Plan
CustomerDetails	Respond Add Customer Request
DeliveryOptionsInformation	Calculate Delivery Time and Price Plan
DeliveryOptionsRequest	Respond Get Delivery Options Request
GetDeliveryInformationMessage	Respond Get Delivery Options Request
KeywordSearchRequest	Show Keyword Search Result Plan
KeywordSearchResponse	Keyword Search Response
NewReleasesRequest	Show New Releases Plan
NewReleasesResponse	Find New Releases Plan
RegisterCustomerResponse	Add New Customer
SessionAccess	
SpecialOffersRequest	Show Special Offers Plan
SpecialOffersResponse	Find Special Offers Plan
SubjectsRequest	Show Subjects Plan
SubjectsResponse	Find Subjects
TopTenBestSellersRequest	Respond Log Out Request
	Show Bookstore Main Page Plan
TopTenBestSellersResponse	Find Top Ten BestSellers Plan
UserDetails	Retrieve Customer Details
UserLoginRequest	Respond SignIn Request
UserLoginResponse	Sign In
WebDispatch	
RegisterCustomerResponse	

Table D.75 Relations identified manually between JACK Event and Prometheus Plan

JACK Event	Prometheus Message	Prometheus Plan
rulePJ33a	Respond Get Delivery Options Request	UserDetails
rulePJ33a	Retrieve Customer Details	UserDetails
rulePJ33a	Respond SignIn Request	UserLoginRequest
rulePJ33a	Sign In	UserLoginResponse
rulePJ33a	Respond Place Order Request	AuthorizationRequest
rulePJ33a	Validate Credit Card	AuthorizationResponse
rulePJ33a	Respond Place Order Request	BookDetailsRequest
rulePJ33a	Respond Place Order Request	BooksByCategoryRequest
rulePJ33a	Check Stock	BookAvalaible
rulePJ33a	Check Stock	BookNotAvalaible
rulePJ33a	Show Advanced Search Result Plan	AdvancedSearchRequest
rulePJ33a	Execute Advanced Search	AdvancedSearchResponse
rulePJ33a	Show Keyword Search Result Plan	KeywordSearchRequest
rulePJ33a	Execute Keyword Search	KeywordSearchResponse
rulePJ33a	Show Books by Category Plan	BooksByCategoryRequest
rulePJ33a	Find Books by Category Plan	BooksByCategoryResponse

rulePJ33a	Show Book Details	BookDetails
rulePJ33a	Show Book Details	BookDetailsRequest
rulePJ33a	Find Book Details Plan	BookDetails
rulePJ33a	Find Book Details Plan	BookDetailsResponse
rulePJ33a	Respond Log Out Request	BestSellersRequest
rulePJ33a	Show Bookstore Main Page Plan	BestSellersRequest
rulePJ33a	Respond Log Out Request	TopTenBestSellersRequest
rulePJ33a	Show Bookstore Main Page Plan	TopTenBestSellersRequest
rulePJ33a	Find Top Ten BestSellers Plan	BestSellersResponse
rulePJ33a	Find Top Ten BestSellers Plan	TopTenBestSellersResponse
rulePJ33a	Show BestSellers Plan	BestSellersRequest
rulePJ33a	Show BestSellers Plan	TopTenBestSellersRequest
rulePJ33a	Find BestSellers Plan	BestSellersResponse
rulePJ33a	Find BestSellers Plan	TopTenBestSellersResponse
rulePJ33a	Show New Releases Plan	NewReleasesRequest
rulePJ33a	Find New Releases Plan	NewReleasesResponse
rulePJ33a	Show Special Offers Plan	SpecialOffersRequest
rulePJ33a	Find Special Offers Plan	SpecialOffersResponse SubjectsRequest
rulePJ33a	Show Subjects Plan	SpecialOffersResponse SubjectsRequest
rulePJ33a	Find Subjects	SubjectsResponse

Table D.76 Relations identified by the tool between Prometheus Message and Prometheus Plan

D.37 JACK Event vs Prometheus Plan (receives)

Table D.77 shows traceability relations identified manually between JACK Event and Prometheus Plan. Table D.78 shows traceability relations identified by the tool between JACK Event and Prometheus Plan. The number of relations identified manually is 34, and the number of relations identified by the tool is 19. Precision and recall calculated is 100% and 55.88%, respectively.

JACK Event	Prometheus Plan
AdvancedSearchRequest	Execute Advanced Search
AdvancedSearchResponse	Show Advanced Search Result Plan
AuthorizationRequest	Validate Credit Card Capability
AuthorizationResponse	Respond Place Order Request
BestSellersRequest	Find BestSellers Plan
BestSellersResponse	Show BestSellers Plan
BookAvalaible	Respond Place Order Request
BookDetails	Respond Add Item to Basket Request
BookDetailsRequest	Find Book Details Plan
BookDetailsResponse	Show Book Details
BookInBasket	Add Item to Basket Plan
BookNotAvalaible	Respond Place Order Request
BookRequired	Check Stock
BooksByCategoryRequest	Find Books by Category Plan
BooksByCategoryResponse	Show Books by Category Plan
CustomerDetails	Add New Customer
DeliveryOptionsInformation	Respond Get Delivery Options Request
DeliveryOptionsRequest	Retrieve Customer Details
GetDeliveryInformationMessage	Calculate Delivery Time and Price Plan
KeywordSearchRequest	Execute Keyword Search
KeywordSearchResponse	Show Keyword Search Result Plan
NewReleasesRequest	Find New Releases Plan
NewReleasesResponse	Show New Releases Plan
RegisterCustomerResponse	Respond Add Customer Request
SessionAccess	
SpecialOffersRequest	Find Special Offers Plan
SpecialOffersResponse	Show Special Offers Plan

SubjectsRequest	Find Subjects
SubjectsResponse	Show Subjects Plan
TopTenBestSellersRequest	Find Top Ten BestSellers Plan
TopTenBestSellersResponse	Respond Log Out Request
TopTenBestSellersResponse	Show Bookstore Main Page
UserDetails	Respond Get Delivery Options Request
UserLoginRequest	Sign In
UserLoginResponse	Respond SignIn Request
WebDispatch	

Table D.77 Relations identified manually between JACK Event and Prometheus Plan

Rule ID	Prometheus Plan	JACK Event
rulePJ33b	Respond Get Delivery Options Request	UserDetails
rulePJ33b	Respond SignIn Request	UserLoginResponse
rulePJ33b	Respond Place Order Request	AuthorizationResponse
rulePJ33b	Show Advanced Search Result Plan	AdvancedSearchResponse
rulePJ33b	Show Keyword Search Result Plan	KeywordSearchResponse
rulePJ33b	Show Books by Category Plan	BooksByCategoryResponse
rulePJ33b	Show Book Details	BookDetails
rulePJ33b	Show Book Details	BookDetailsResponse
rulePJ33b	Respond Log Out Request	BestSellersResponse
rulePJ33b	Show Bookstore Main Page Plan	BestSellersResponse
rulePJ33b	Respond Log Out Request	TopTenBestSellersResponse
rulePJ33b	Show Bookstore Main Page Plan	TopTenBestSellersResponse
rulePJ33b	Show BestSellers Plan	BestSellersResponse
rulePJ33b	Show BestSellers Plan	TopTenBestSellersResponse
rulePJ33b	Show New Releases Plan	NewReleasesResponse
rulePJ33b	Show Special Offers Plan	SpecialOffersResponse SubjectsRequest
rulePJ33b	Show Subjects Plan	SubjectsResponse

Table D.78 Relations identified by the tool between Prometheus Plan and JACK Event

D.38 JACK Event vs Prometheus Message

Table D.79 shows traceability relations identified manually between JACK Event and Prometheus Message. Table D.80 shows traceability relations identified by the tool between JACK Event and Prometheus Message. The number of relations identified manually is 33, and the number of relations identified by the tool is 36. The number of relations identified correctly is 26 and the number of relation missing is 10. The number of relations identified incorrectly by the tool is 10. Precision and recall calculated is 72.22% and 78.78%, respectively.

JACK Event	Prometheus Message
AdvancedSearchRequest	AdvancedSearch Request
AdvancedSearchResponse	Advanced Search Response
AuthorizationRequest	Authorization Request
AuthorizationResponse	Authorization Response
BestSellersRequest	BestSellers Request
BestSellersResponse	BestSeller Response
BookAvalaible	Book Avalaible
BookDetails	Add to Basket Response
BookDetailsRequest	Book Details Request
BookDetailsResponse	Book Details Response
BookInBasket	Add to Basket Request

BookNotAvalaible	Book Not Avalaible
BookRequired	Book Request
BooksByCategoryRequest	Books by Category Request
BooksByCategoryResponse	Books by Category Response
CustomerDetails	Add Customer Request
DeliveryOptionsInformation	Delivery Time and Price Response
DeliveryOptionsRequest	User Details Request
GetDeliveryInformationMessage	Delivery Time and Price Request
KeywordSearchRequest	Keyword Search Request
KeywordSearchResponse	Keyword Search Response
NewReleasesRequest	New Releases Request
NewReleasesResponse	New Releases Response
RegisterCustomerResponse	Add Customer Response
SessionAccess	
SpecialOffersRequest	Special Offers Request
SpecialOffersResponse	Special Offers Response
SubjectsRequest	Subjects Request
SubjectsResponse	Subjects Response
TopTenBestSellersRequest	Top Ten BestSellers Request
TopTenBestSellersResponse	Top Ten BestSellers Response
UserDetails	User Details Response
UserLoginRequest	User Login Request
UserLoginResponse	User Login Response
WebDispatch	
	WebSession Request

Table D.79 Relations between JACK Event and Prometheus Message

Rule ID	Prometheus Message	JACK Event
rulePJ1a	User Details Request	UserDetails
rulePJ1a	User Details Response	UserDetails
rulePJ1a	User Login Request	UserLoginRequest
rulePJ1a	User Login Response	UserLoginResponse
rulePJ1a	Authorization Request	AuthorizationRequest
rulePJ1a	Authorization Response	AuthorizationResponse
rulePJ1a	Book Request	BookDetailsRequest
rulePJ1a	Book Request	BooksByCategoryRequest
rulePJ1a	Book Avalaible	BookAvalaible
rulePJ1a	Book Avalaible	BookNotAvalaible
rulePJ1a	Book Not Avalaible	BookAvalaible
rulePJ1a	Book Not Avalaible	BookNotAvalaible
rulePJ1a	Advanced Search Request	AdvancedSearchRequest
rulePJ1a	Advanced Search Response	AdvancedSearchResponse
rulePJ1a	Keyword Search Request	KeywordSearchRequest
rulePJ1a	Keyword Search Response	KeywordSearchResponse
rulePJ1a	Books by Category Request	BooksByCategoryRequest
rulePJ1a	Books by Category Response	BooksByCategoryResponse
rulePJ1a	Book Details Request	BookDetails
rulePJ1a	Book Details Request	BookDetailsRequest
rulePJ1a	Book Details Response	BookDetails
rulePJ1a	Book Details Response	BookDetailsResponse
rulePJ1a	Top Ten BestSellers Request	BestSellersRequest
rulePJ1a	Top Ten BestSellers Request	TopTenBestSellersRequest
rulePJ1a	Top Ten BestSellers Response	BestSellersResponse
rulePJ1a	Top Ten BestSellers Response	TopTenBestSellersResponse
rulePJ1a	BestSellers Request	BestSellersRequest
rulePJ1a	BestSellers Request	TopTenBestSellersRequest
rulePJ1a	BestSellers Response	BestSellersResponse
rulePJ1a	BestSellers Response	TopTenBestSellersResponse
rulePJ1a	New Releases Request	NewReleasesRequest
rulePJ1a	New Releases Response	NewReleasesResponse
rulePJ1a	Special Offers Request	SpecialOffersRequest
rulePJ1a	Special Offers Response	SpecialOffersResponse
rulePJ1a	Subjects Request	SubjectsRequest
rulePJ1a	Subjects Response	SubjectsResponse

Table D.80 Relations between Prometheus Message and JACK Event

Appendix E – Introduction to BDI architecture

This appendix gives an introduction to the BDI architecture. Initially, we present different types of agent architecture used to build multi-agents and then we describe in detail the BDI architecture that was used by our research.

E.1 Agent Architectures

Agent architectures can be general classified in three types: deliberative architectures, reactive architectures, hybrid architectures.

Reactive architectures do not maintain a symbolic representation of the environment and actions are performed using rules. Agents are situated in the environment and perceive the environment. Depending of the event that occurs in the environment a rule is executed and actions are performed.

In the deliberative architecture, a symbolic representation of the environment is created and the agent performs actions to manipulate these symbols. The actions performed are based on logical reasoning using theorem provers [Genesereth1987]. The drawback of this architecture is that it is difficult to represent real world using a symbolic representation and that logic reasoning to determine what action to perform is a very resource and time consuming task. Several multi-agent systems use a deliberative architecture to support reasoning and some of them are based on the BDI architecture [Bratman1988]. BDI architectures have been proposed to address the problem of resource boundedness. In our research, we focus on the BDI architecture. The three main reasons that we choose the BDI architecture is because they are based on folk philosophy to explain rational reasoning (therefore it is easier to understand the main concepts), has been formalised by logic theories (that can be used to demonstrate programs), and it has been implemented several times (e.g. PRS, JACK, and JADEX). We give more details about the BDI architecture in the next section.

Hybrid architectures combine deliberative and reactive behaviour. Examples of hybrid architectures are: TouringMachines, and INTERRRAP [Luck2004].

E.2 BDI Architecture

The BDI architecture is based on the philosophy theory proposed by Bratman [Bratman1988]. Bratman explain human rational action in terms of beliefs, desires and intentions. Beliefs represent information that we have about the environment, desires represent the state of affairs that we want to achieve and intentions represent desires that we had committed to achieve.

Accordingly to Bratman human practical reasoning is divided in two parts. First, we decide what we want to achieve (*desires*) and second we decide how to achieve our desires. The process to decide how to achieve our desires is decomposed in two parts. First we generate plans choosing a sequence of actions from a set of possible actions then we select what plan to execute based on our beliefs. For instance, I decide that I want go to see a movie at Leicester Square after work. The first thing that I should do is to select what movie I want to see and then to choose what cinema and time I want to see the movie. I buy a ticket to see the movie at 6.30pm. I still have to decide how to go from my work to Leicester Square. My possible actions are cycle, walk, take the underground or take the bus. I would like go to the cinema by walking, but I have to finish writing a document by today. I don't know what time I can leave the work and it is raining, so I wait until the end of the day to take my decision how to go to the cinema.

Bratman also points out to the problem of resource *boundeness* that means that the making decision process of humans or computers takes place under limited amount of time. Suppose that I finish writing my document at 6.00pm. I go to a Journey Planner system to check how long takes to go from my work to the cinema. It takes thirteen minutes cycling, twenty and nine minutes using the tube, fifty minutes by bus and fourteen minutes walking. It is raining so I decide to take the tube. There is no much time between the time that I had finished writing my document and the time that I have to take the tube. If I take too much time deliberating how to go to the cinema, to take the tube it would be not anymore an option to be considered because it would be too late to arrive on time.

Bratman also declares that intentions are persistent. If when I arrive at the tube station I realise that there are some delays on the trains. I will not give up on my intention to watch the movie. For instance, I can re-consider to cycle or select a new plan such as to take a taxi. To consider when and how agents should drop intentions Rao describes in [Rao1991] three strategies: blind

commitment, single-minded commitment and open-minded commitment. In blind commitment an agent continues to maintain the intention until it has been achieved. When an agent uses the single-minded commitment it will continue to maintain the intention until it has been achieved or it is impossible to achieve. In the open-minded commitment, an agent will maintain the intention until it has been achieved or the intention still compatible with others intentions.

The Figure E.1 shows a generic BDI architecture. A BDI agent is situated in an environment, receive inputs from the environment (events) and perform actions (output). An agent is composed of beliefs, desires, intentions and one interpreter. The interpreter continually selects what desires to commit (i.e. create intentions), select options how to achieve the desire, execute the option selected, drop successful and impossible intentions, perceive the environment and perform actions.

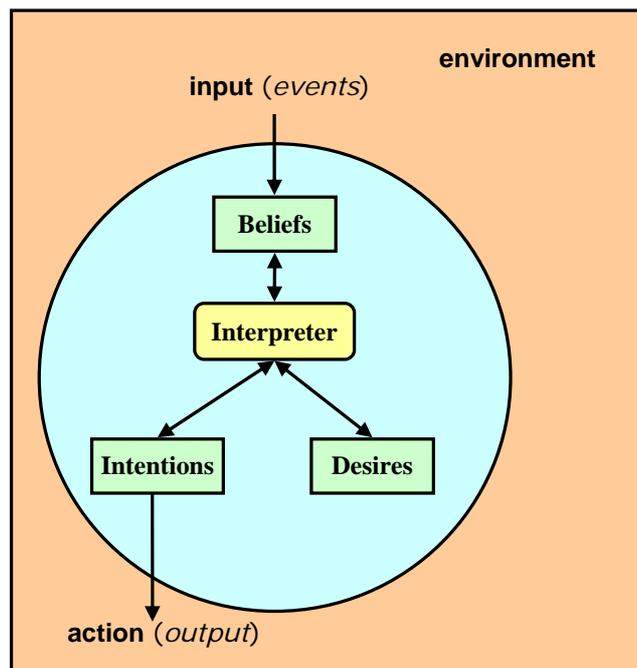


Figure E.1 A generic BDI architecture

The BDI model has been implemented several times. Examples of implementation are PRS, dMARS, JAM, JACK, and Jadex.

Appendix F - Traceability Relations between i^* and Prometheus

This appendix describes traceability relations between i^* and Prometheus elements

We have identified seven different types of traceability relations between the various elements in the models used in our approach. The types of traceability relations are *overlaps*, *contributes to*, *uses*, *creates*, *achieves*, *depends on*, and *composed of*. We present below descriptions of these different types of relations.

- **Overlaps** – in this type of relation, an element e1 overlaps with an element e2 (an element e2 overlaps with an element e1), if e1 and e2 refer to elements with common aspects of the agent software development. As shown in Tables F.1 and F.2, an *overlaps* relation may hold between a) goal in Prometheus and SD goal in i^* ; b) goal in Prometheus and a SD task; c) an agent in Prometheus and an actor in i^* d) a percept in Prometheus and a SD resource in i^* ; d) a message in Prometheus and a SD resource in i^* ; e) a goal in Prometheus and a SR goal in i^* ; f) a goal in Prometheus and a SR task in i^* ; g) an action in Prometheus and a SR task in i^* ; h) a data in Prometheus and a SR Resource in i^* .
- **Contributes (Contributed by)** - in this type of relation, an element e1 contributes to an element e2, if e1 helps to achieve or realise another element e2. As shown in Tables F.1 and F.2, a *contributes* relation may hold between a) role in Prometheus and a SD goal in i^* ; b) a role in Prometheus and a SD Task in i^* ; c) a role in Prometheus and actor in i^* ; d) a capability in Prometheus and a SD goal in i^* ; e) a capability in Prometheus and a SD task in i^* ; f) a data in Prometheus and a SD goal in i^* ; g) a data in Prometheus and a SD task in i^* ; h) a capability in Prometheus and a SR goal in i^* .
- **Uses (Used by)** - in this type of relation, an element e1 uses an element e2, if e1 requires the existence of e2 in order to achieve its objective. As shown in Tables F.1 and F.2, a *contributes* relation may hold between a) a role in Prometheus and a SD

- resource in i^* ; b) an agent in Prometheus and a SD resource in i^* ; c) a capability in Prometheus and a SD resource in i^* ; d) a plan in Prometheus and a SD resource in i^* ; e) a data in Prometheus and an actor in i^* ; f) a role in Prometheus and a SR resource in i^* ; g) an agent in Prometheus and a SR goal in i^* ; h) a capability in Prometheus and a SR resource in i^* ; i) a plan in Prometheus and a SR goal in i^* ; j) a data in Prometheus and a SR resource in i^* .
- **Creates (Created by)** - in this type of relation an element e1 creates an element e2, if e1 generates element e2. As shown in Tables F.1 and F.2, a *creates* relation may hold between a) a plan in Prometheus and an actor in i^* ; b) role in Prometheus and SR resource in i^* ; c) an agent in Prometheus and a SR resource in i^* ; d) a capability in Prometheus and a SR resource in i^* ; e) a plan in Prometheus and a SR resource in i^* ; f) a scenario in Prometheus and a SR resource in i^* .
 - **Achieves (Achieved by)** - in this type of relation an element e1 achieves an element e2, if e1 meets the expectations and needs of e2. As shown in Tables F.1 and F.2, a *achieves* relation may hold between a) an agent in Prometheus and a SD goal in i^* ; b) an agent in Prometheus and a SD task in i^* ; c) a plan in Prometheus and a SD goal in i^* ; d) a plan in Prometheus and a SD task in i^* ; e) a role in Prometheus and a SR goal in i^* ; f) a role in Prometheus and a SR task in i^* ; g) an agent in Prometheus and a SR task in i^* ; h) a capability in Prometheus and a SR task in i^* ; i) a plan in Prometheus and and a SR task in i^* .
 - **Depends on (Is Dependent)** - in this type of relation an element e1 depends on an element e2, if the existence of e1 relies on the existence of e2, or if changes in e2 have to be reflected in e1. As shown in Tables F.1 and F.2, a *depends* relation may hold between a) a goal in Prometheus and an actor in i^* ; b) a scenario in Prometheus and a SD goal in i^* ; c) a scenario in Prometheus and an actor in i^* .
 - **Composed of** - in this type of relation and element e1 is composed of an element e2, if e1 is a complex element formed by element e2. As shown in Tables F.1 and F.2, a *depends* relation may hold between a) a capability in Prometheus and an actor in i^* ; b) a scenario in Prometheus and a SD resource; c) a scenario in Prometheus and a SR goal

in i^* ; d) a scenario in Prometheus and a SR task in i^* ; e) a scenario in Prometheus and a SR goal in i^* ; f) a scenario in Prometheus and a SR task in i^* .

Tables F.1 and F.2 present the different types of traceability relations for the main types of elements in i^* SD model and Prometheus models, and i^* SR model and Prometheus models, respectively. In Tables F.1 and F.2, apart from overlaps relations that are bi-directional, the direction of a relation is represented from a row [i] to a column [j] (e.g. “Prometheus role contributes to SD goal”). We do not consider i^* soft goals in Tables F.1 and F.2 since soft goals are concerned with non-functional aspects of a system while elements in Prometheus are concerned with functional aspects of a system. We define below the various types of traceability relations and give some examples from the perspective of each specific pair of artefacts that are associated.

i^* Prometheus	SD Goal	SD Resource	SD Task	Actor
Goal	Overlaps	---	Overlaps	Is Dependent
Role	Contributes to	Uses	Contributes to	Contributes to
Agent	Achieves	Uses	Achieves	Overlaps
Capability	Contributes to	Uses	Contributes to	Compose
Plan	Achieves	Uses	Achieves	Created by
Percept	---	Overlaps	---	---
Data	Contributes to	---	Contributes to	Is Used
Scenario	Depends on	Composed of	Depends on	Is Dependent
Message	---	Overlaps	---	---

Table F.1 Relations between Prometheus and i^* SD

i^* Prometheus	SR Goal	SR Resource		SR Task
Goal	Overlaps	---		Overlaps
Role	Achieves	Uses	Creates	Achieves
Agent	Achieves	Uses	Creates	Achieves
Capability	Contributed by	Uses	Creates	Achieves
Plan	Achieves	Uses	Creates	Achieves

Action	---	---		Overlaps
Data	Used by	Overlaps		Used by
Scenario	Composed of	Uses	Creates	Composed of

Table F.2 Relations between Prometheus and *iSR elements**

- Prometheus Goal vs SD Goal – A goal g_1 in Prometheus has an *overlaps* traceability relation with a SD goal g_2 in *i** if the name of the goal g_1 is synonyms to the name of the goal g_2 and the number of sub-elements of the Prometheus goal g_1 that is similar to the sub-goals and sub-tasks of the goal g_2 is greater than a threshold (e.g. 40%) or the number of sub-elements of the Prometheus goal g_1 that is similar to the sub-goals and sub-tasks of the goal g_2 is greater than a threshold (e.g. 60%). For instance, *Browse Book* SD goal in *i** has a synonyms name to *Browse book* goal in Prometheus (see Figure F.1). *Browse Book* SD goal is decomposed on *Browse By Special Offer*, *Browse By BestSeller*, *Browse By Category*, *Browse by New Releases* sub-goals and *Browse book* goal is decomposed on *Browse By category*, *Browse by new release*, *Browse by bestseller*, and *Browse by special offer* sub-tasks. The degree of similarity between the sub-elements of *Browse Book* SD goal and *Browse Book* Prometheus goal is equal to 100% because *Browse By Special Offer*, *Browse By BestSeller*, *Browse By Category*, *Browse by New Releases* sub-tasks are synonyms to *Browse by category*, *Browse by new release*, *Browse by bestseller*, *Browse by special offer* sub-goals, respectively. Therefore there is a traceability relation between *Browse Book* Prometheus goal and *Browse book* SD goal.

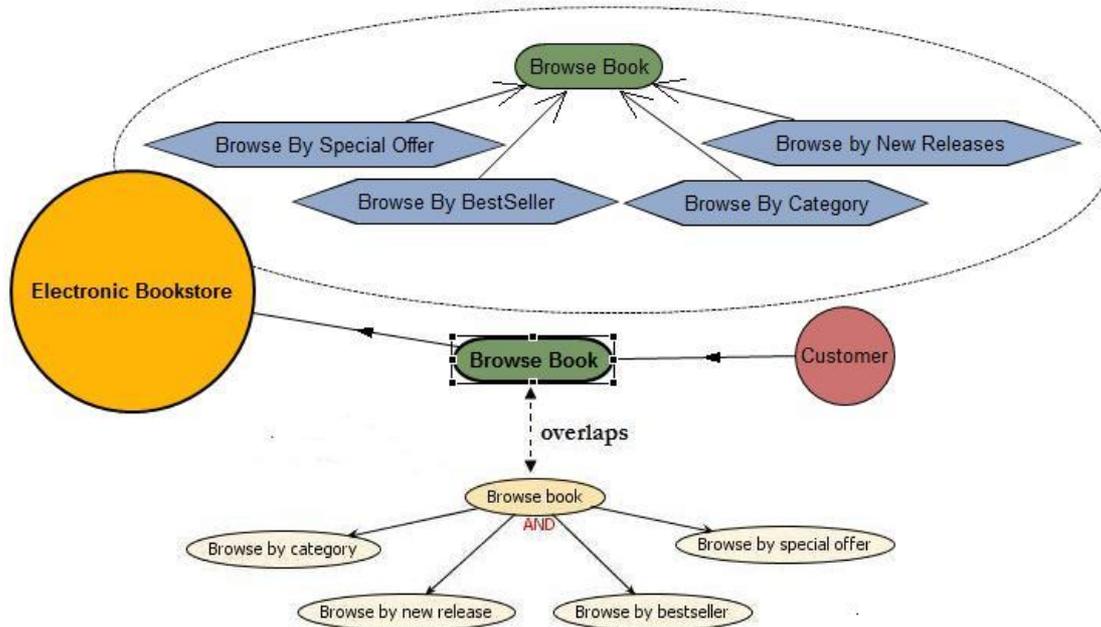


Figure F.1 Prometheus Goal vs SD Goal overlaps dependency

- Prometheus Goal vs SD Task – A goal g_1 in Prometheus has an *overlaps* traceability relation with a SD task t_1 in i^* if the name of the goal g_1 is synonyms to the name of the task t_2 and the number of sub-elements of the Prometheus goal g_1 that is similar to the sub-goals and sub-tasks of the task t_1 is greater than a threshold (e.g. 40%) or the number of sub-elements of the Prometheus goal g_1 that is similar to the sub-goals and sub-tasks of the task t_1 is greater than a threshold (e.g. 60%). For instance, *Monitor Shipment* SD task in i^* (see Figure F.2) has a synonyms name to *Monitor delivery* goal in Prometheus (see Figure F.3). *Monitor Shipment* SD task is decomposed in *Determine Delivery Status Location*, *Update Delivery Status*, and *Log Delivery Problem* sub-tasks and *Monitor delivery* Prometheus goal is decomposed in *Determine delivery status*, *Update delivery status* and *Log delivery problems*. The degree of similarity between the sub-elements of *Find Best Land Time for an Aircraft* SD goal and *Find Best Land Time for an Aircraft* Prometheus goal is equal to 50%, *Query Best Landing Time from All Runway Manager* sub-task is synonyms to *Query Best Landing Time from All Runway Manager* sub-goal. Therefore there is a traceability relation between *Find Best Land Time for an Aircraft* Prometheus goal and *Find Best Landing Time for an Aircraft* SD goal.

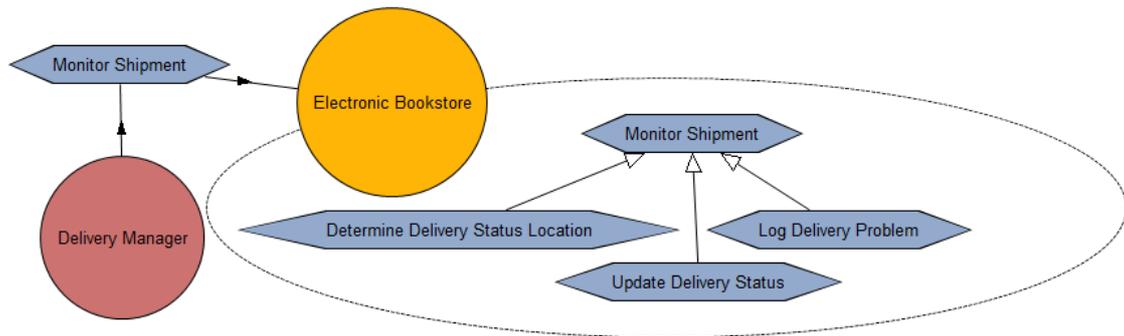


Figure F.2 Monitor Shipment task dependency

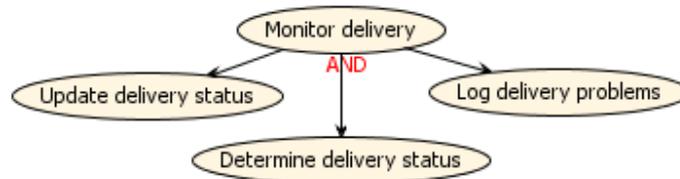


Figure F.3 Monitor delivery goal in Prometheus

- Prometheus Goal vs Actor – A goal g_1 in Prometheus has a *depends on* traceability relation with an actor a_1 in i^* when the goal g_1 has an *overlaps* traceability relation with a goal g_2 in i^* and the actor depends on that goal g_2 . For instance, Browse Book SD goal has an *overlaps* traceability relation with Browse book goal in Prometheus (see Figure F.4). Therefore, an *depends on* traceability relation is created between the Customer actor in i^* and browse book goal in Prometheus.

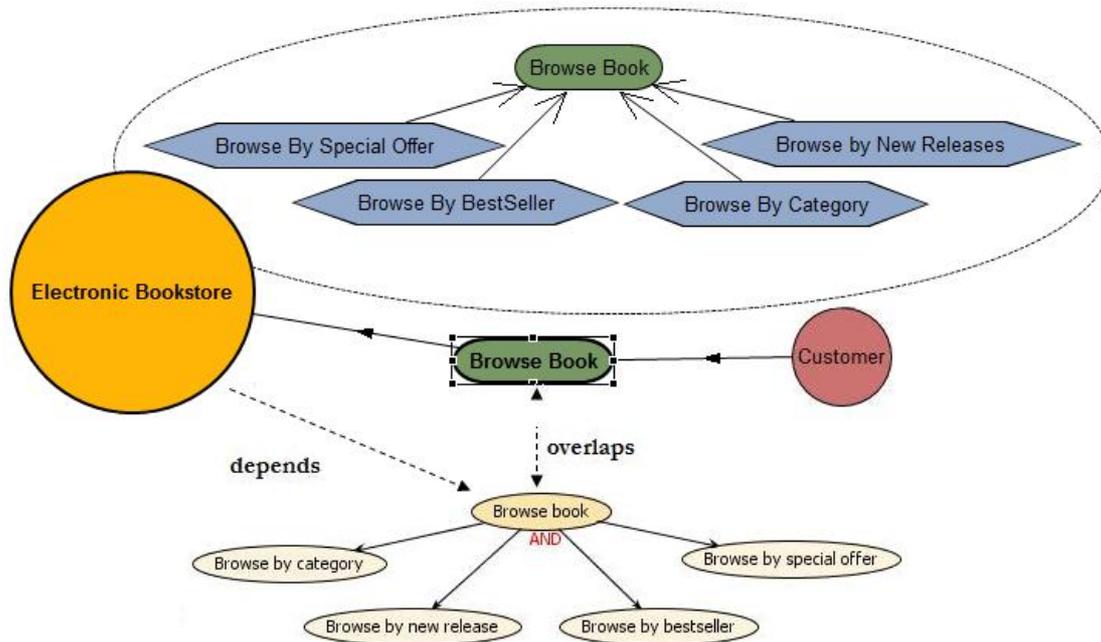


Figure F.4 Prometheus Goal vs Actor depends on traceability relation

- Prometheus Goal vs SR Goal – A goal g_1 in Prometheus has an *overlaps* traceability relation with a SR goal g_2 in i^* if the name of the goal g_1 is synonyms to the name of the goal g_2 and the number of sub-elements of the Prometheus goal g_1 that is similar to the sub-goals and sub-task of the goal g_2 is greater than a threshold (e.g. 40%) or the number of sub-elements of the Prometheus goal g_1 that is similar to the sub-goals and sub-tasks of the goal g_2 is greater than a threshold (e.g. 60%). For instance, *Browse Book* SR goal in i^* has a synonyms name to *Browse book* goal in Prometheus (see Figure F.5). *Browse Book* SR goal is decomposed on *Browse By Special Offer*, *Browse By BestSeller*, *Browse By Category*, *Browse by New Releases* sub-tasks and *Browse book* goal is decomposed on *Browse By category*, *Browse by new release*, *Browse by bestseller*, and *Browse by special offer* sub-goals. The degree of similarity between the sub-elements of *Browse Book* SR goal and *Browse Book* Prometheus goal is equal to 100% because *Browse By Special Offer*, *Browse By BestSeller*, *Browse By Category*, *Browse by New Releases* sub-goals are synonyms to *Browse by category*, *Browse by new release*, *Browse by bestseller*, *Browse by special offer* sub-tasks, respectively. Therefore there is a traceability relation between *Browse Book* Prometheus goal and *Browse book* SR goal.

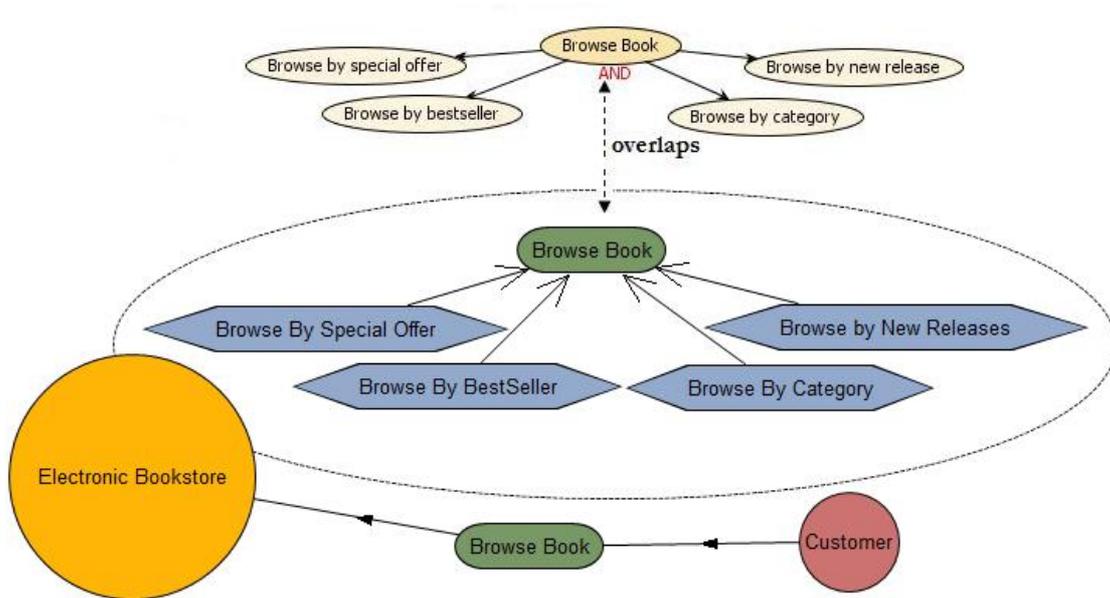


Figure F.5 Prometheus Goal vs SR Goal overlaps traceability relation

- Prometheus Goal vs SR Task – A goal g_1 in Prometheus has an *overlaps* traceability relation with a SR task t_1 in i^* if the name of the goal g_1 is synonyms to the name of the task t_1 and the number of sub-elements of the Prometheus goal g_1 that is similar to the sub-goals and sub-task of the task t_1 is greater than a threshold (e.g. 40%) or the number of sub-elements of the Prometheus goal g_1 that is similar to the sub-goals and sub-tasks of the task t_1 is greater than a threshold (e.g. 60%). For instance, *Organize Delivery* SR task in i^* has a synonyms name to *Arrange delivery* goal in Prometheus (see Figure F.6). *Organize Delivery* SR task is decomposed on *Obtain Delivery Options*, *Compute Delivery Time Estimates*, *Place Delivery Request* sub-tasks and *Arrange delivery* goal is decomposed on *Log outgoing delivery*, *Calculate delivery time estimates*, and *Get delivery options*. *Get delivery options* is synonyms to *Obtain Delivery Options* and *Calculate delivery time estimates* is synonyms to *Compute Delivery Time Estimates*. The percentage of sub-goals of the *Arrange delivery* goal that is similar to the sub-tasks or sub-goals of the *Organize Delivery* is 66.7% that is greater than the threshold of 60%. Therefore, there is an overlaps traceability relation between *Arrange delivery* Prometheus goal and *Organize Delivery* SR task.

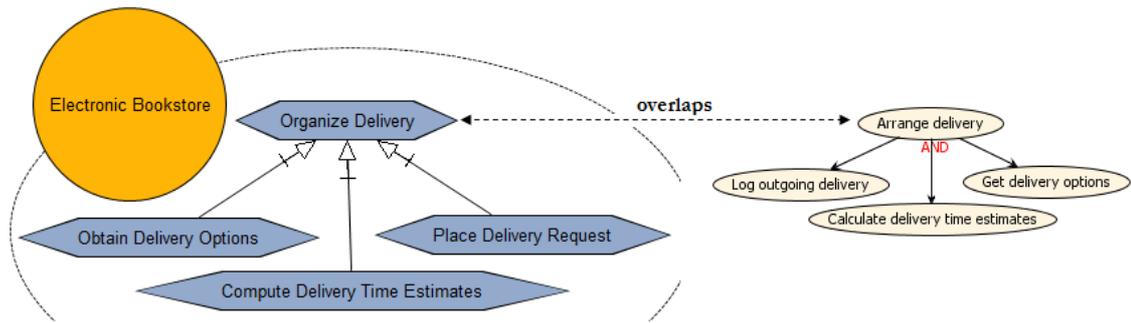


Figure F.6 Prometheus Goal vs SR Task overlaps traceability relation

- Prometheus Role vs SD Goal – a Prometheus Role r_1 in Prometheus has a *contributes* traceability relation with a SD Goal g_1 when the role includes a goal g_2 that has an *overlaps* traceability relation with the goal g_1 . For instance, *Book Finding* role includes *Browse book* goal that has an *overlaps* traceability relation with Browse book SD Goal (see Figure F.7). Therefore, there is a *contributes* traceability relation between *Browse Book* SD Goal and *Book Finding* role.

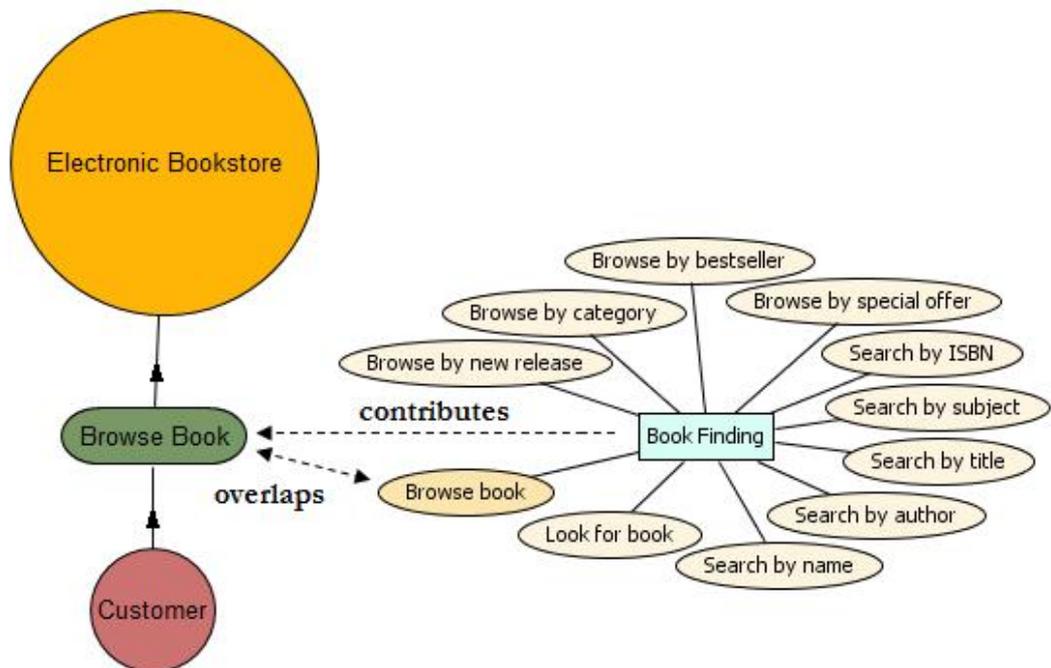


Figure F.7 Prometheus Role vs SD Goal uses traceability relation

- Prometheus Role vs SD Resource - a Prometheus Role r_1 has an *uses* traceability relation with a SD Resource r_2 when the role r_1 includes a percept p_1 that has an *overlaps*

traceability relation with the SD Resource r_1 in i^* . For instance, *Purchasing* role in Prometheus includes *Credit Card Details* percept that has an *overlaps* traceability with *Credit Card Details* SD resource (see Figure F.8). Therefore, there is an *uses* traceability relation between *Purchasing* role and *Credit Card Details* SD resource.

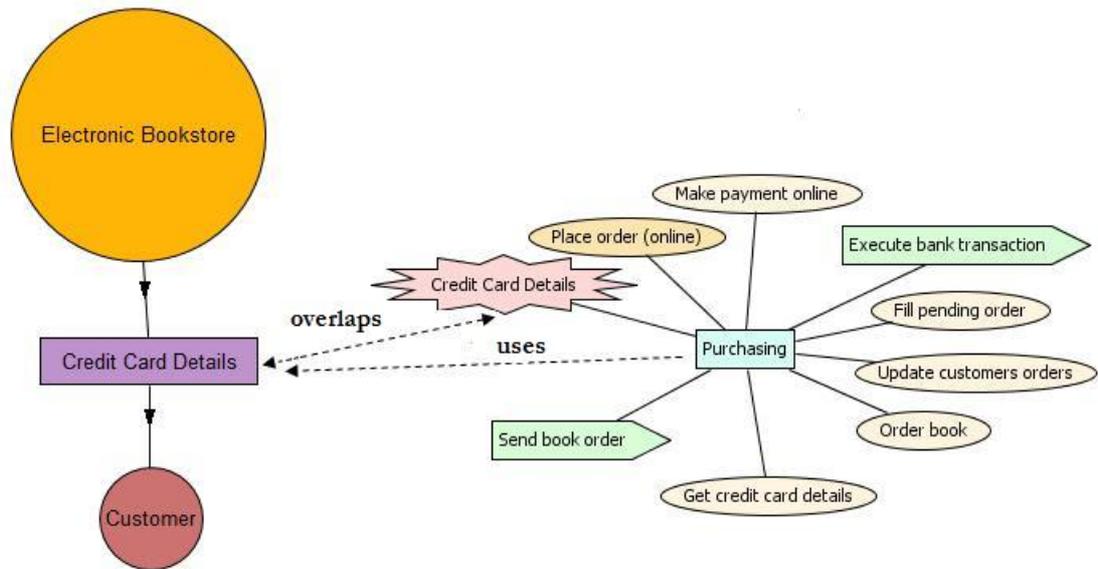


Figure F.8 Prometheus Role vs SD Resources uses relation

- Prometheus Role vs SD Task – a Prometheus Role r_1 in Prometheus has a *contributes* traceability relation with a SD Task t_1 when the role includes a goal g_1 that has an *overlaps* traceability relation with the task t_1 . For instance, Delivery Management role in Prometheus includes Monitor delivery goal that has an overlaps traceability relation with Monitor Shipment SD task (see Figure F.9). Therefore, there is a contributes traceability relation between Delivery Management and Monitor Shipment.

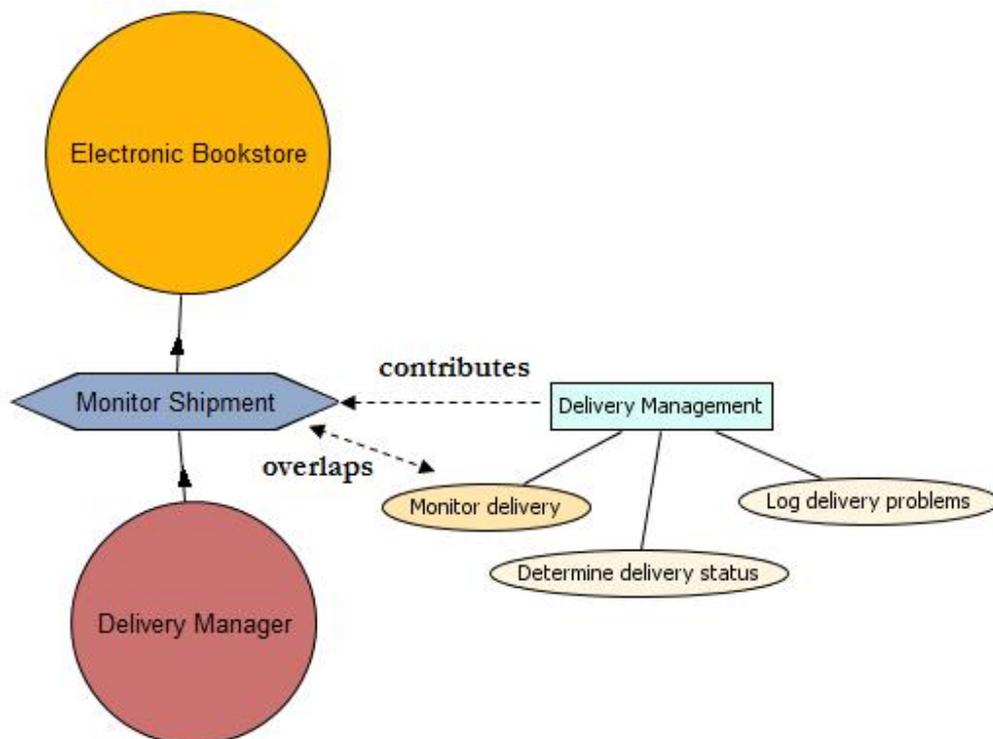


Figure F.9 Prometheus Role vs SD Task contributes relation

- Prometheus Role vs Actor – role r_1 in Prometheus has a *contributes to* traceability relation with an actor a_1 in i^* when there is an *overlaps* traceability relation between the actor and an agent a_2 and the agent a_2 includes the role r_1 . For instance, the Stock Manager agent in Prometheus has an overlaps traceability relation with Stock Manager actor in i^* and the Stock Manager agent plays the Stock Management role (see Figure F.10). Therefore, there is a contributes traceability relation between Stock Management role in Prometheus and Stock Manager actor in i^* .

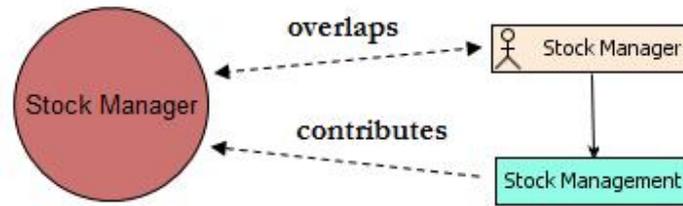


Figure F.10 Prometheus Role vs Actor contributes relation

- Prometheus Role vs SR Goal – a Prometheus Role r_1 in Prometheus has an *achieves* traceability relation with a SR Goal g_1 when the role includes a goal g_2 that has an *overlaps* traceability relation with the goal g_1 . For instance, the Book Finding role achieves the Browse book goal in Prometheus and the Browse book goal in Prometheus has an overlaps traceability relation with Browse Book SR goal in i^* (see Figure F.11). Therefore, there is an achieves traceability relation between Book Finding role in Prometheus and Browse Book SR Goal in i^* .

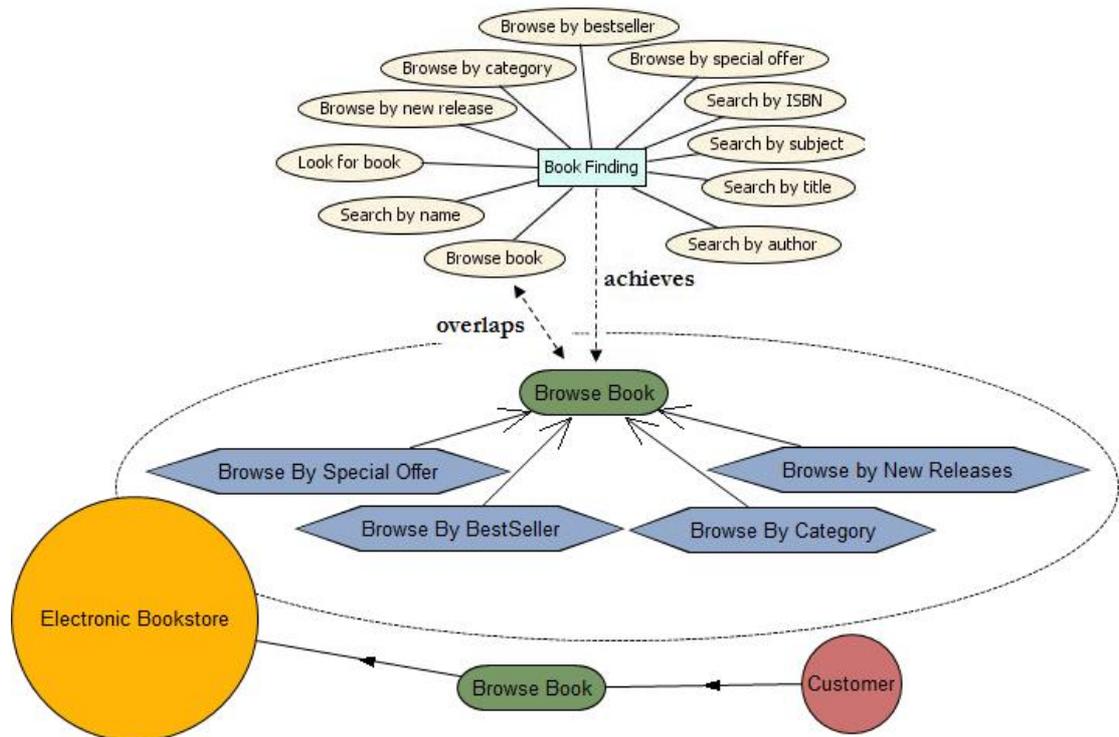


Figure F.11 Prometheus Role vs SR Goal achieves traceability relation

- Prometheus Role vs SR Resource (*uses*) – a Prometheus Role r_1 in Prometheus has an *uses* traceability relation with a SR Resource r_1 when the role uses a data d_1 that has an *overlaps* traceability relation with the SR Resource r_1 in i^* . For instance, Delivery Handling role uses CourierDB data in Prometheus and CourierDB data has an *overlaps* traceability relation with Courier DB SR data in i^* (see Figure F.12). Therefore, there is an *uses* traceability relation between Courier DB SR data and CourierDB data in Prometheus.

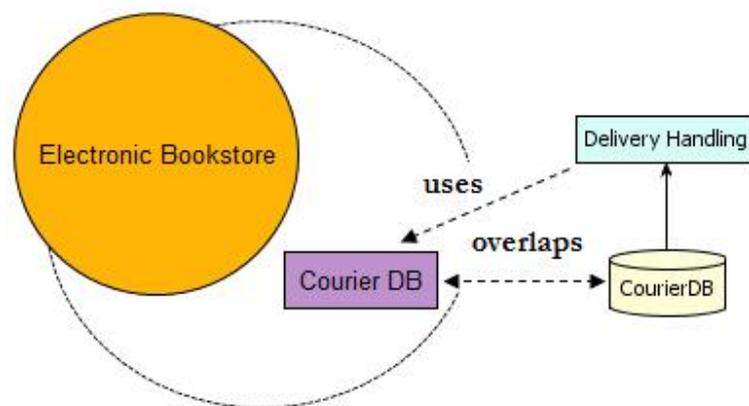


Figure F.12 Prometheus Role vs SR Resource uses relation

- Prometheus Role vs SR Resource (*creates*) – a Prometheus Role r_1 in Prometheus has *creates* traceability relation with a SR Resource r_1 when the role produces a data d_1 that has an *overlaps* traceability relation with the SR Resource r_1 in i^* . For instance, Purchasing role produces Customer Order data in Prometheus and Customer Order SR resource in i^* has an *overlaps* traceability relation with Customer Order data (see Figure F.13). Therefore, there is a *creates* traceability relation between Purchasing role and Customer Order data.

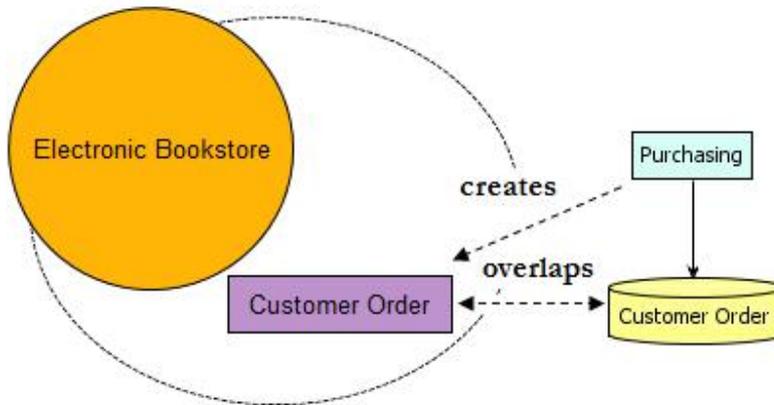


Figure F.13 Prometheus Role vs SR Resource creates relation

- Prometheus Role vs SR Task (*achieves*) - a role r_1 in Prometheus has an *achieves* traceability relation with a SR task t_1 when the SR task t_1 has an *overlaps* traceability relation with a goal g_1 in Prometheus and the role r_1 achieves the goal g_1 . For instance, Delivery Handling role achieves Arrange delivery goal in Prometheus and Organize Delivery SR task in i^* has an overlaps traceability relation with Arrange delivery goal (see Figure F.14). Therefore, there is an achieves traceability relation between Delivery Handling role and Organize Delivery SR task.

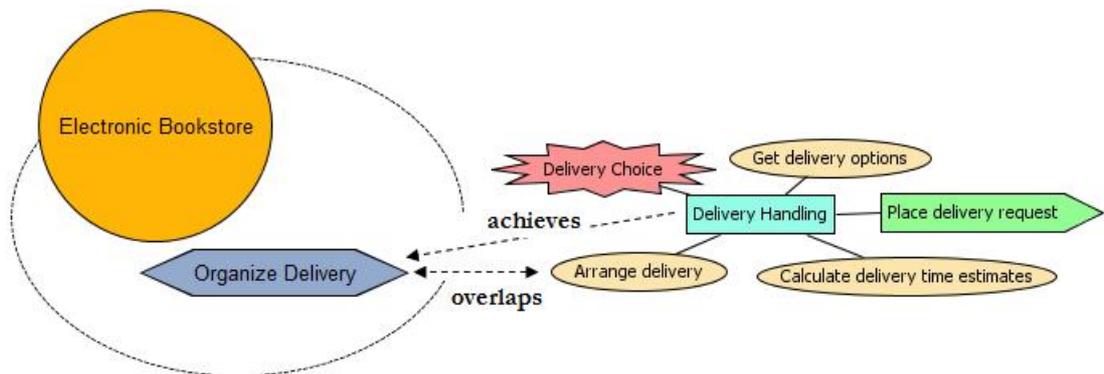


Figure F.14 Prometheus Role vs SR Task achieves traceability relation

- Prometheus Agent vs SD Goal – an agent a_1 in Prometheus has an achieves traceability relation with a SD Goal g_1 when the Prometheus agent a_1 achieves a goal g_2 and the goal g_2 has an overlaps traceability relation with the goal g_1 . For instance, Stock Manager agent achieves Browse book goal in Prometheus and Browse book goal

in Prometheus has an overlaps traceability relation with Browse Book SD Goal (see Figure F.15). Therefore, there is a achieves traceability relation between Stock Manager agent in Prometheus and Browse Book SD goal in i^* .

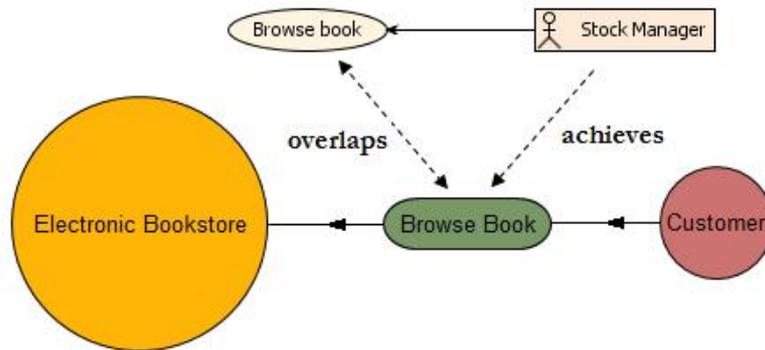


Figure F.15 Prometheus Agent vs SD Goal achieves traceability relation

- Prometheus Agent vs SD Resource – an agent a_i in Prometheus has an *uses* traceability relation with a SD Resource r_i in i^* when the agent receives a message m_i that has an *overlaps* traceability relation with the SD Resource r_i or when the agent receives a percept p_i that has an overlaps traceability relation with the SD Resource r_i . For instance, Stock Manager agent receives Keyword Search percept in Prometheus and Keyword Search SD Resource has an overlaps traceability relation with Keyword Search percept (see Figure F.16). Therefore, there is an uses traceability relation between Stock Manager agent in Prometheus and Keyword Search SD Resource in i^* .

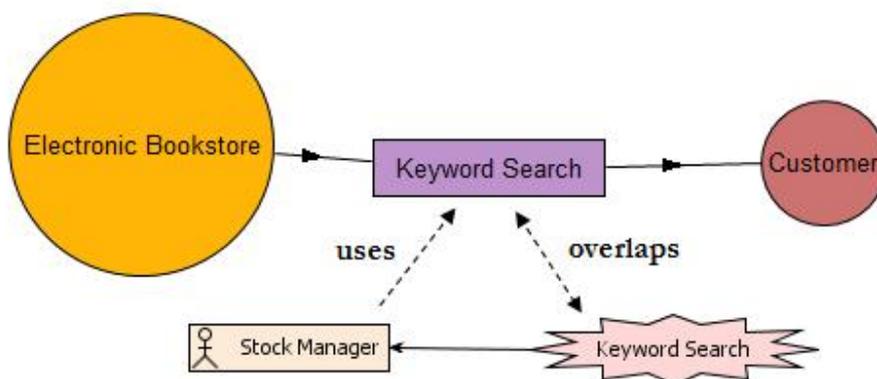


Figure F.16 Prometheus Agent vs SR Resource uses traceability relation

- Prometheus Agent vs SD Task – an agent a_1 in Prometheus has an *achieves* traceability relation with a SD Task t_1 when the Prometheus agent a_1 includes a goal g_2 that has an *overlaps* traceability relation with the SD Task in i^* . For instance, Delivery Manager agent in Prometheus achieves Monitor delivery goal in Prometheus and Monitor delivery goal in Prometheus has an overlaps traceability relation with Monitor Shipment SD task (see Figure F.17). Therefore, there is an achieves traceability relation between Delivery Manager agent in Prometheus and Monitor Shipment SR task in i^* .

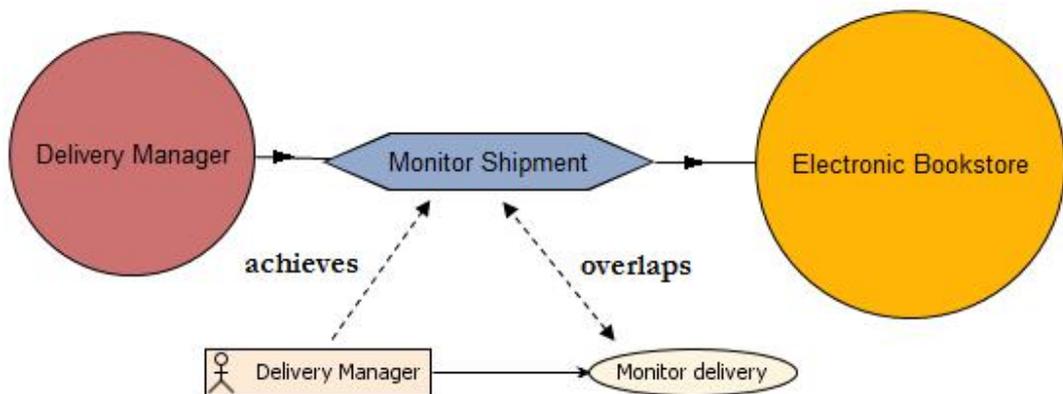


Figure F.17 Prometheus Agent vs SD Task achieves traceability relation

- Prometheus Agent vs Istar Actor – an agent a_1 in Prometheus has an *overlaps* traceability relation with an actor in i^* when the name of the agent in Prometheus is synonyms to the name of actor in i^* . For instance, the name of the *Delivery Manager* actor in i^* is synonyms to the name of the *Delivery Manager* agent in Prometheus (see Figure F.18). Therefore, there is a overlaps traceability relation between *Delivery Manager* actor in i^* and *Delivery Manager* agent in Prometheus.

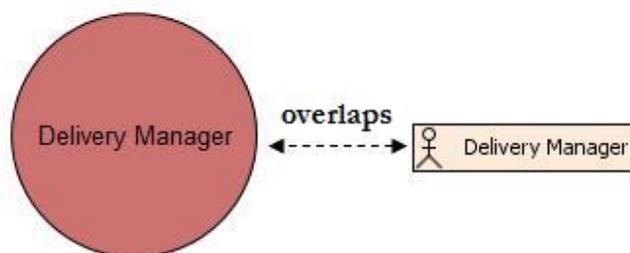


Figure F.18 Prometheus Agent vs Istar Actor overlaps traceability relation

- Prometheus Agent vs SR Goal – an agent a_1 in Prometheus has an achieves traceability relation with a SR Goal g_1 in i^* when the agent a_1 achieves a goal g_1 in Prometheus that has an *overlaps* traceability relation with the SR Goal g_1 in i^* . For instance, Stock Manager agent achieves Browse book goal in Prometheus and Browse book goal in Prometheus has an overlaps traceability relation with Browse Book SR goal in i^* (see Figure F.19). Therefore, there is an achieves traceability relation between Stock Manager agent in Prometheus and Browse Book SR goal in i^* .

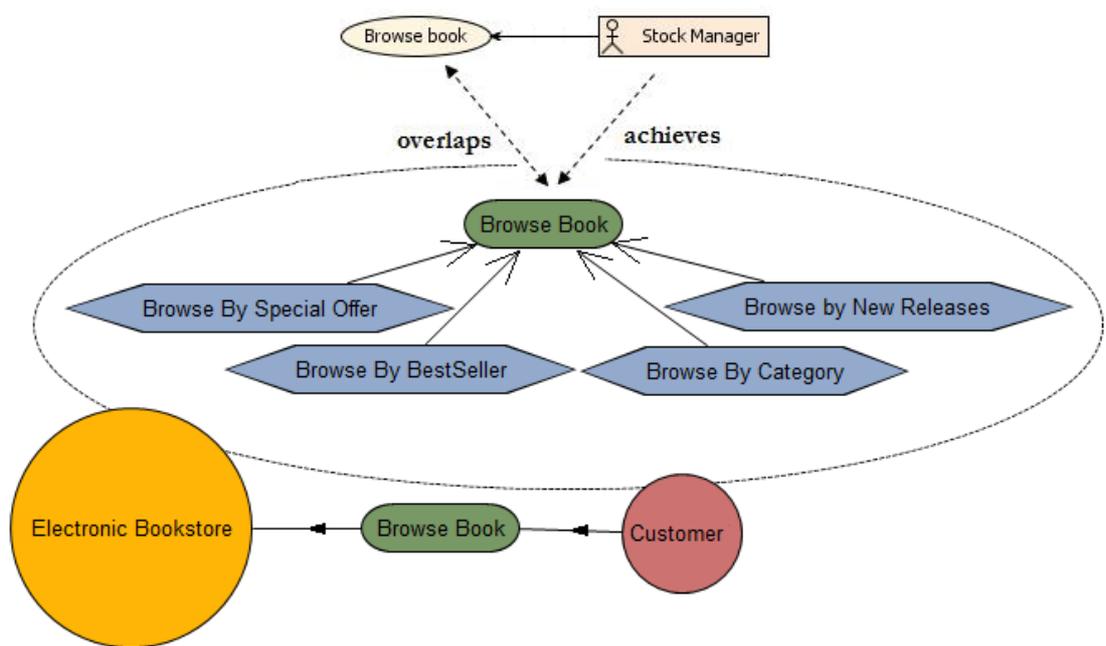


Figure F.19 Prometheus Agent vs SR Goal achieves traceability relation

- Prometheus Agent vs SR Resource (uses) – an agent a_1 in Prometheus has a *uses* traceability relation with a SR Resource r_1 in i^* when the Agent reads a data in Prometheus that has an *overlaps* traceability relation with the SR Resource r_1 . For instance, Delivery Manager agent uses Couier data that has an *overlaps* traceability relation with Courier DB SR Resource (see Figure F.20). Therefore, there is a *uses* traceability relation between *Delivery Manager* agent and *Courier DB* SR Resource in i^* .

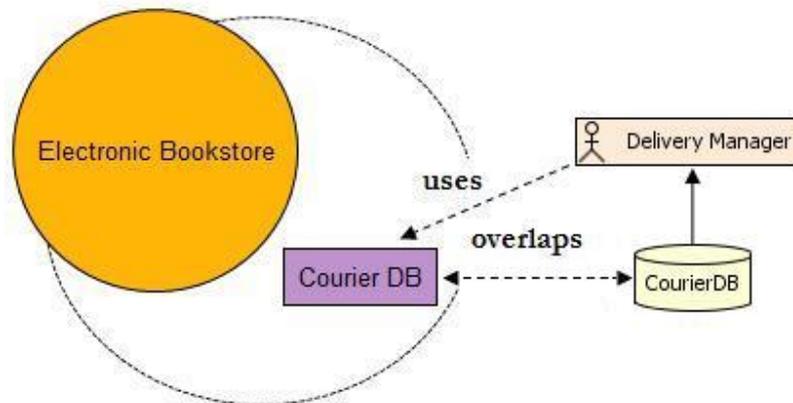


Figure F.20 Prometheus Agent vs SR Resource uses traceability relation

- Prometheus Agent vs SR Resource (creates) – an agent a_1 in Prometheus has a *creates* traceability relation with a SR Resource r_1 in i^* when the Agent writes on data in Prometheus that has an *overlaps* traceability with the SR Resource r_1 . For instance, Stock Manager agent writes on Customer Order data and Customer Order data has an overlaps traceability relation Customer SR Resource (see Figure F.21). Therefore, there is a creates traceability relation between Stock Manager agent in Prometheus and Customer Order SR Resource in i^*

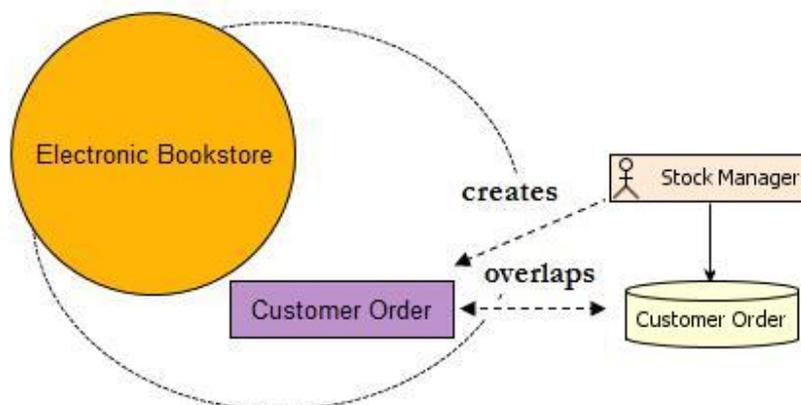


Figure F.21 Prometheus Agent vs SR Resource creates traceability relation

- Prometheus Agent vs SR Task – an agent a_i in Prometheus has an *achieves* traceability relation with a SR Task t_1 in i^* if an agent a_i includes a goal g_1 that has an *overlaps* traceability relation with the task t_1 . For instance, Delivery Manager agent in Prometheus achieves Arrange delivery goal in Prometheus and Organize Delivery SR task has an overlaps traceability relation with Arrange delivery goal in Prometheus (see Figure F.22). Therefore, there is an achieves traceability relation between Delivery Manager agent in Prometheus and Organize Delivery SR task.

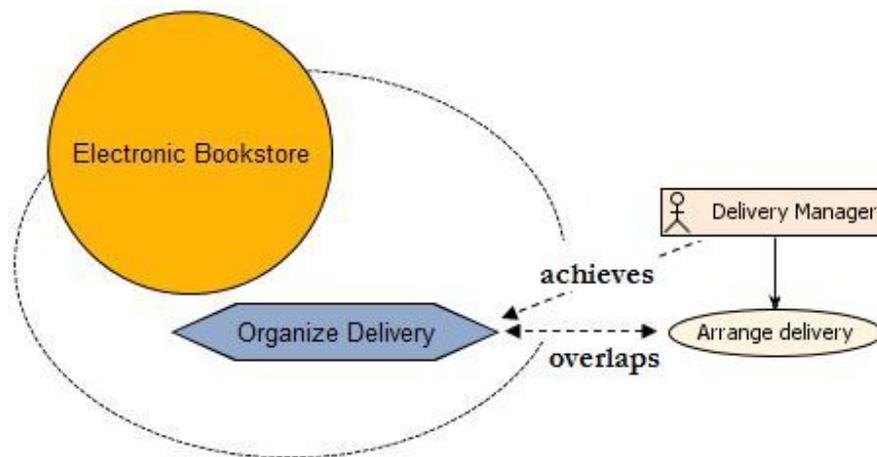


Figure F.22 Prometheus Agent vs SR Task achieves traceability relation

- Prometheus Capability vs SD Goal – a capability c_1 in Prometheus has a *contributes* traceability relation with a SD Goal g_1 when the capability c_1 includes a plan p_1 that achieves a goal g_1 that has an *overlaps* traceability relation with the SD Goal g_1 . For instance, the *Arrival Sequencing* capability includes *Request Slot Plan* plan in Prometheus. The Request Slot Plan plan achieves *Allocate Runway Slot* goal in Prometheus (see Figure F.23). Allocate Runway Slot goal in Prometheus has an overlaps traceability relations with Allocate Runway Slot SD goal. Therefore, there is a contributes traceability relation between Arrival Sequencing capability and Allocate Runway Slot SR plan.

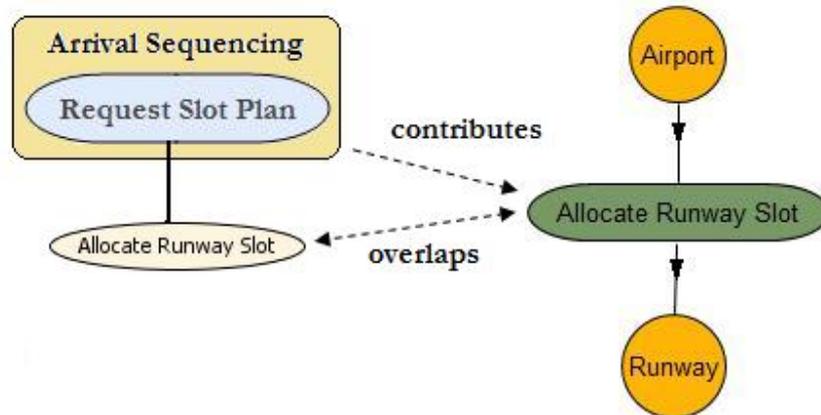


Figure F.23 Prometheus Capability vs SD Goal contributes traceability relation

- Prometheus Capability vs SD Resource – a capability c_1 in Prometheus has a *uses* traceability relation with a SD Resource r_1 in i^* when the capability receives a message m_1 that has an *overlaps* traceability relation with the SD Resource r_1 , or when the capability receives a percept p_1 that has an *overlaps* traceability relation with the SD Resource r_1 . For instance, Arrival Sequencing capability receives Aircraft message (see Figure F.24). AircraftEvent message has an *overlaps* traceability relation with Slot Allocated SD resource. Therefore, there is an *uses* traceability relation between ArrivalSequencing capability and Slot Allocated SD resource.

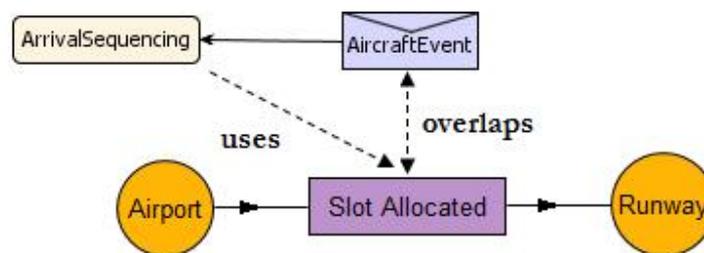


Figure F.24 Prometheus Capability vs SD Resource uses traceability relation

- Prometheus Capability vs SD Task – a capability c_1 in Prometheus has a *contributes* traceability relation with a SD task t_1 when the capability c_1 includes a plan p_1 that achieves a goal g_1 that has an *overlaps* traceability relation with the SD Task t_1 . For instance, Delivery Monitoring capability includes Monitor Delivery Plan that achieves Monitor delivery (see Figure F.25). Monitor delivery goal in Prometheus has an *overlaps* traceability relation with Monitor Shipment SD task.

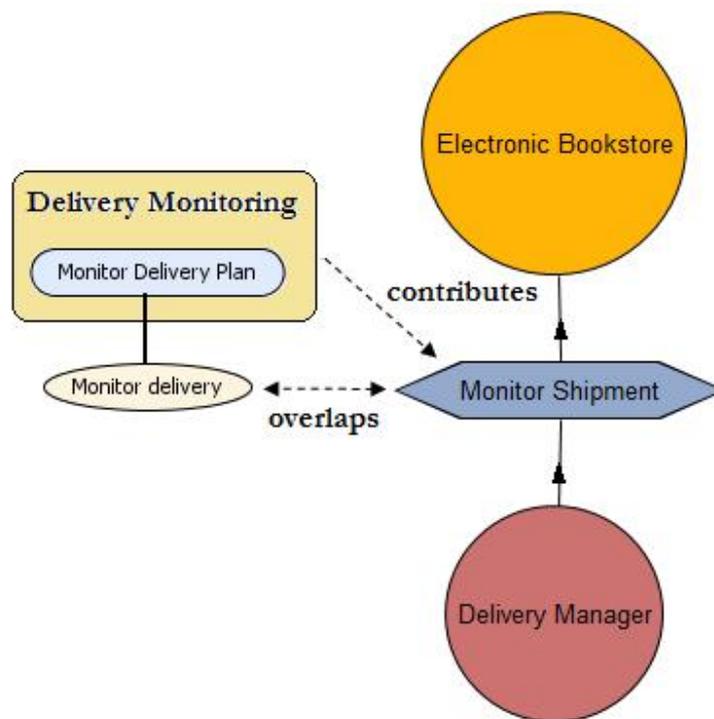


Figure F.25 Prometheus Capability vs SD Task contributes traceability relation

- Prometheus Capability vs Actor – an actor a_1 in i^* has a *composed of* traceability relation with a capability c_1 when there is an *overlaps* traceability relation between the actor a_1 and an agent a_1 and the agent a_1 includes the capability c_1 . For instance, the *Airport* actor in i^* has an *overlaps* traceability relation with the *Airport* agent in Prometheus and the *Airport* agent includes *ArrivalSequencing* capability (see Figure F.26). Therefore, there is a *composed of* traceability relation between *Airport* actor and *ArrivalSequencing* capability.

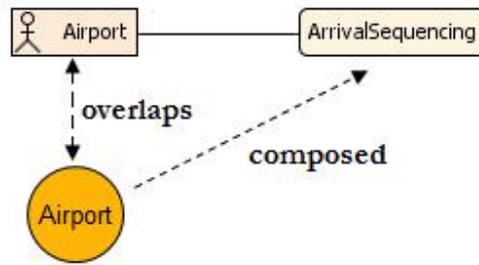


Figure F.26 Prometheus Capability vs Actor composed relation

- Prometheus Capability vs SR Goal - a capability c_1 in Prometheus has a *contributes* traceability relation with a SR Goal g_1 when the capability c_1 includes a plan p_1 that achieves a goal g_1 that has an *overlaps* traceability relation with the SR Goal g_1 . For instance, *Runway Assigning* capability in Prometheus includes *Runway Assign* plan that achieves *Allocate Runway Slot* goal (see Figure F.27). The *Allocate Runway Slot* goal in Prometheus has an *overlaps* traceability relation with *Allocate Runway Slot* SR Goal in \mathcal{I}^* . Therefore there is a *contributes* traceability relation between *Runway Assigning* capability and *Allocate Runway Slot* SR Goal in \mathcal{I}^* .

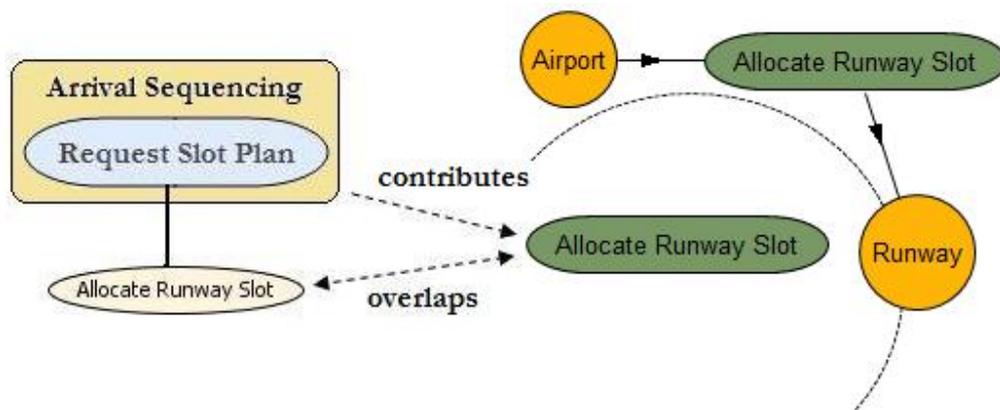


Figure F.27 Prometheus Capability vs SR Goal contributes traceability relation

- Prometheus Capability vs SR Resource (uses) – a capability c_1 in Prometheus has an *uses* traceability relation with a SR Resource r_1 in \mathcal{I}^* when the Capability reads a data in Prometheus that has an *overlaps* traceability with the SR Resource r_1 . For instance, *Flying* capability reads *Landing Information* data that has an *overlaps* traceability relation with *Landing Information* SR Resource (see Figure F.28). Therefore, there is an *uses* traceability relation between *Landing Information* SR Resource and *Flying* capability.

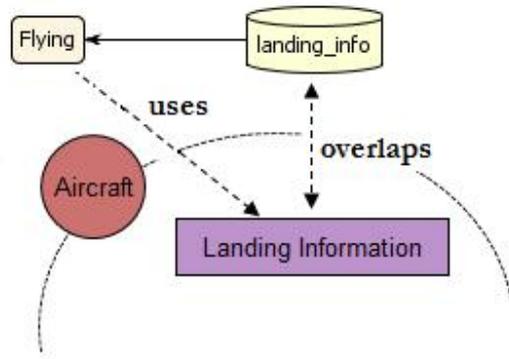


Figure F.28 Prometheus Capability vs SR Resource uses traceability relation

- Prometheus Capability vs SR Resource (creates) – a capability c_1 in Prometheus has a *creates* traceability relation with a SR Resource r_1 in i^* when the Capability writes on data in Prometheus that has an *overlaps* traceability with the SR Resource r_1 (see Figure F.29). For instance, *Flying* capability writes *Landing Information* data that has an *overlaps* traceability relation with *Landing Information* SR Resource. Therefore, there is a *creates* traceability relation between *Landing Information* SR Resource and *Flying* capability.

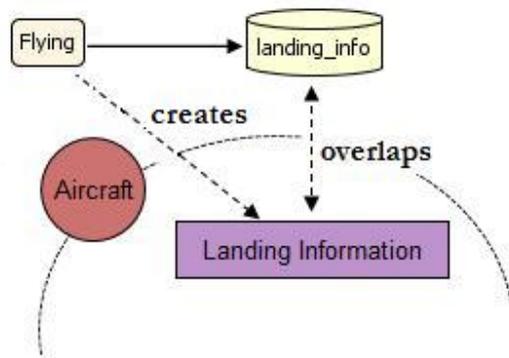


Figure F.29 Prometheus Capability vs SR Resource creates traceability relation

- Prometheus Capability vs SR Task – a capability c_1 in Prometheus has an *achieves* traceability relation with a SR Task t_1 when the capability c_1 includes a plan p_1 that achieves a goal g_1 that has an *overlaps* traceability relation with the SR Task t_1 . For instance, the *Flying* capability in Prometheus includes *Initiate Approach* plan that achieves *Initiate Aircraft Approach* goal (see Figure F.30). The *Initiate Aircraft Approach* goal in Prometheus has an *overlaps* traceability relation with *Initiate Approach* SR Task in i^* .

Therefore there is an overlaps traceability relation between *Flying* capability and *Initiate Approach* SR Task in i^* .

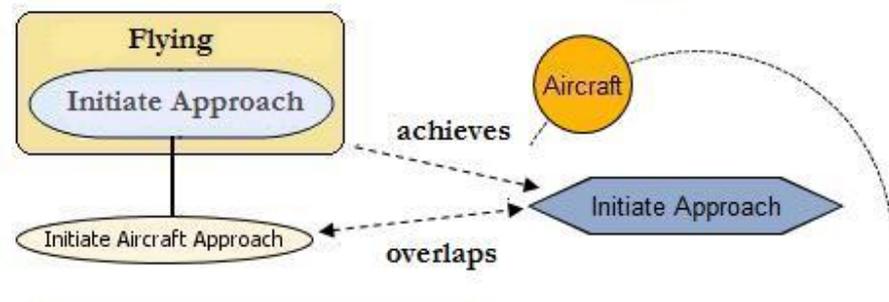


Figure F.30 Prometheus Capability vs SR Resource uses traceability relation

- Prometheus Plan vs SD Goal – a plan p_1 in Prometheus has an achieves traceability relation with the SD Goal g_1 when the plan p_1 achieves a goal g_2 in Prometheus and there is an *overlaps* traceability relation between the SD Goal g_1 and the goal g_2 in Prometheus. For instance, RequestSlot plan in Prometheus achieves Allocate Runway Slot goal and Allocate Runway Slot goal in Prometheus has an overlaps traceability relation with Allocate Runway Slot SD goal in i^* (see Figure F.31). Therefore, there is an achieves traceability relation between RequestSlot plan in Prometheus and Allocate Runway Slot SD Goal.

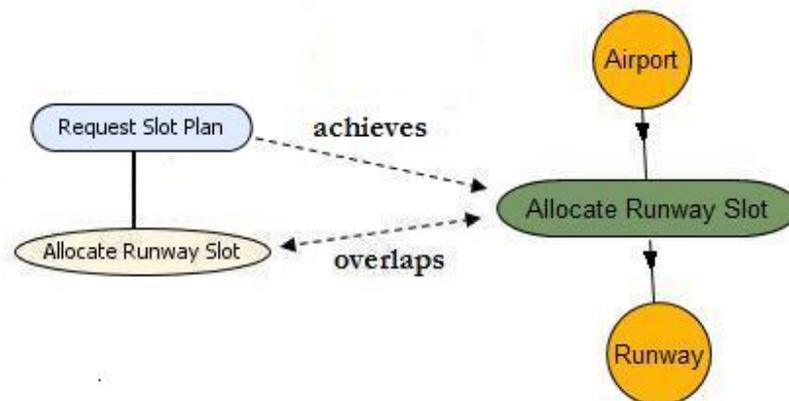


Figure F.31 Prometheus Plan vs SD Goal contributes traceability relation

- Prometheus Plan vs SD Resource – a plan p_1 in Prometheus has an *uses* traceability relation with the SD Resource r_1 when there is a plan p_1 that receives a message m_1 that has an *overlaps* traceability relation with the resource r_1 . For instance, RequestSlot plan in Prometheus receives AircraftEvent message (see Figure F.32). AircraftEvent message in Prometheus has an overlaps traceability relation with Slot Allocated SD resource. Therefore, there is an uses traceability relation between RequestSlot plan and Slot Allocated SD resource.

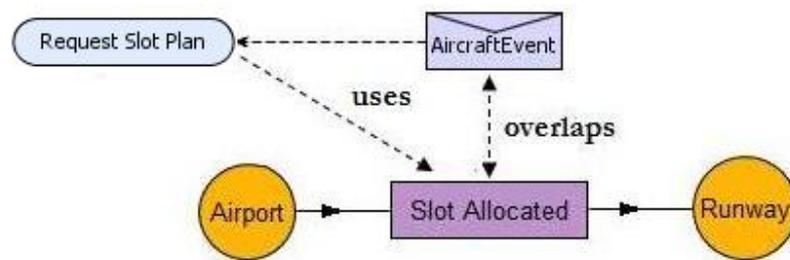


Figure F.32 Prometheus Plan vs SD Resource uses traceability relation

- Prometheus Plan vs SD Task - a plan p_1 in Prometheus has an achieves traceability relation with the SD task t_1 when the plan p_1 achieves a goal g_1 in Prometheus and there is an *overlaps* traceability relation between the SD task t_1 and the goal g_1 in Prometheus. For instance, Monitor Delivery Plan plan in Prometheus achieves Monitor delivery goal in Prometheus and Monitor delivery goal in Prometheus has an overlaps traceability relation with Monitor Shipment SD task (see Figure F.33). Therefore, there is an achieves traceability relation between Monitor Delivery Plan plan and Monitor Shipment SD task in i^* .

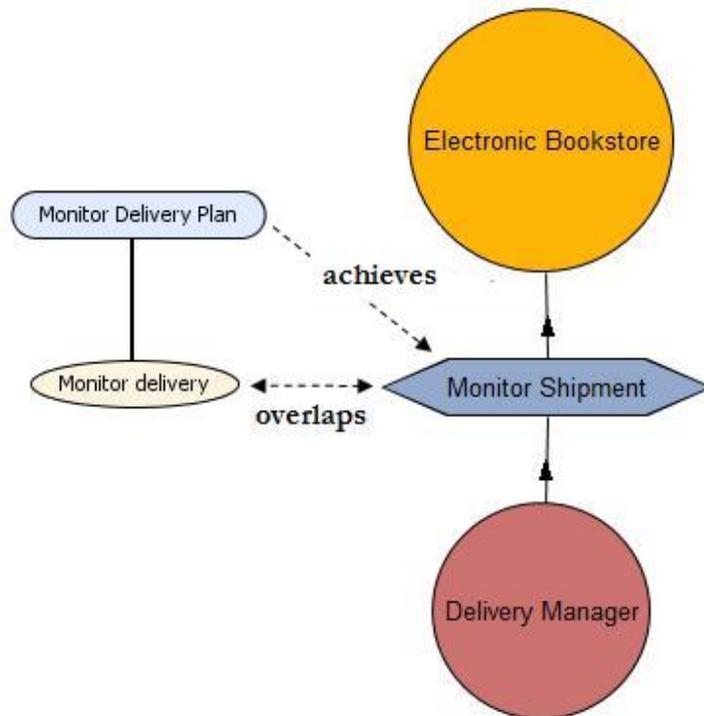


Figure F.33 Prometheus Plan vs SD Task achieves traceability relation

- Prometheus Plan vs Actor – a plan p_1 in Prometheus has a creates traceability relation with an actor a_1 when the actor a_1 contains a SR task t_1 or SR goal g_1 that has an *overlaps* traceability relation with a goal g_1 that the plan p_1 achieves or when the actor a_1 satisfies a goal dependency or task dependency where the goal or task have an *overlaps* traceability relation with the Prometheus goal g_1 . For instance, Runway actor satisfies Allocate Runway Slot goal dependency and RequestSlot plan in Prometheus achieves Allocate Runway Goal that has an *overlaps* traceability relation with Allocate Runway Slot goal (see Figure F.34). Therefore, there is a creates traceability relation between Runway actor in i^* and RequestSlot plan in Prometheus.

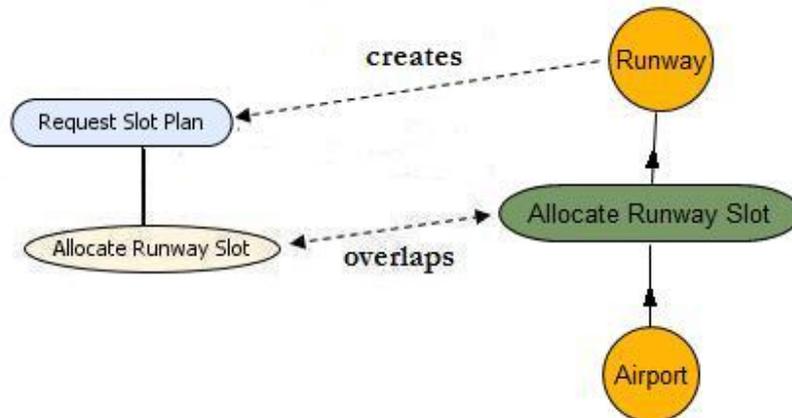


Figure F.34 Prometheus vs Actor creates traceability relation

- Prometheus Plan vs SR Goal – a plan p_1 in Prometheus has an *achieves* traceability relation with a SR Goal g_1 in i^* when the plan p_1 achieves a goal g_1 that has *overlaps* traceability relation with the SR Goal g_1 . For instance, Request Slot Plan plan in Prometheus achieves Allocate Runway Slot goal and Allocate Runway Slot goal in Prometheus has an overlaps traceability relation with Allocate Runway Slot SR goal in i^* (see Figure E.35). Therefore, there is an achieves traceability relation between Request Slot Plan plan in Prometheus and Allocate Runway Slot SR goal.

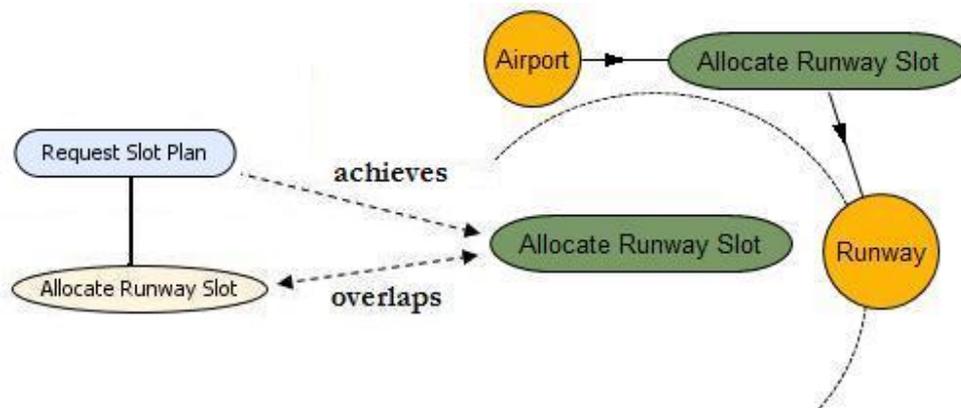


Figure F.35 Prometheus Plan vs SR Goal achieves traceability relation

- Prometheus Plan vs SR Resource (uses) – a plan p_1 in Prometheus has an *uses* traceability relation with a SR Resource r_1 in i^* when the plan p_1 reads a data d_1 that has an *overlaps* traceability relation with the SR Resource r_1 . For instance, *Monitor Aircraft*

plan reads Landing Information data and *Landing Information* SR Resource has an *overlaps* traceability relation with *Landing Information* data in Prometheus (see Figure F.36). Therefore, there is an *uses* traceability relation between *Monitor Aircraft* plan and *Landing Information* SR Resource.

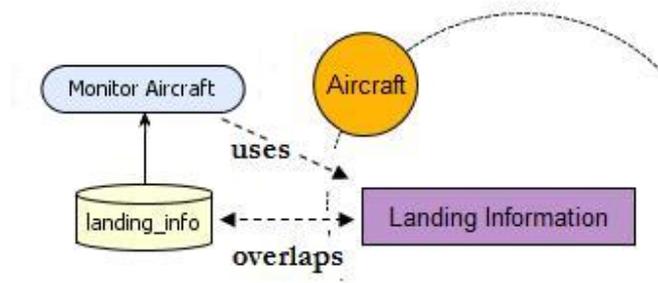


Figure F.36 Prometheus Plan vs SR Resource uses traceability relation

- Prometheus Plan vs SR Resource (creates) - a plan p_1 in Prometheus has a *creates* traceability relation with a SR Resource r_1 in \mathcal{I}^* when the plan p_1 writes a data d_1 that has an *overlaps* traceability relation with the SR Resource r_1 . For instance, Assign Slot Plan writes *landing_info* data that has an *overlaps* traceability relation with *Landing Information* SR resource (see Figure F.37). Therefore, there is a *creates* traceability relation between Assign Slot Plan plan in Prometheus and *Landing Information* SR resource.

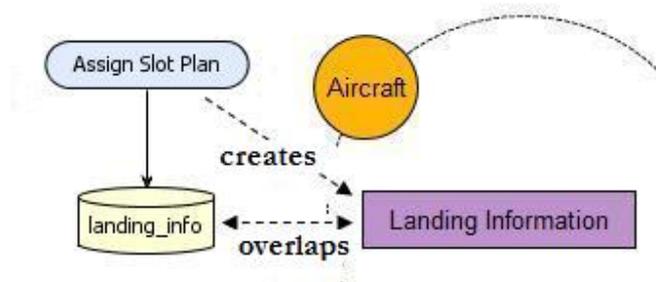


Figure F.37 Prometheus Plan vs SR Resource creates traceability relation

- Prometheus Plan vs SR Task – a plan p_1 in Prometheus has an *achieves* traceability relation with a SR Task t_1 in \mathcal{I}^* when the plan p_1 achieves a goal g_1 that has *overlaps* traceability relation with the SR Task t_1 . For instance, Initiate Approach plan achieves

Initiate Aircraft Approach goal and Initiate Aircraft Approach goal has an overlaps traceability relation with Initiate Approach SR task (see Figure F.38). Therefore, there is an achieves traceability relation between Initiate Approach plan in Prometheus and Initiate Approach SR task

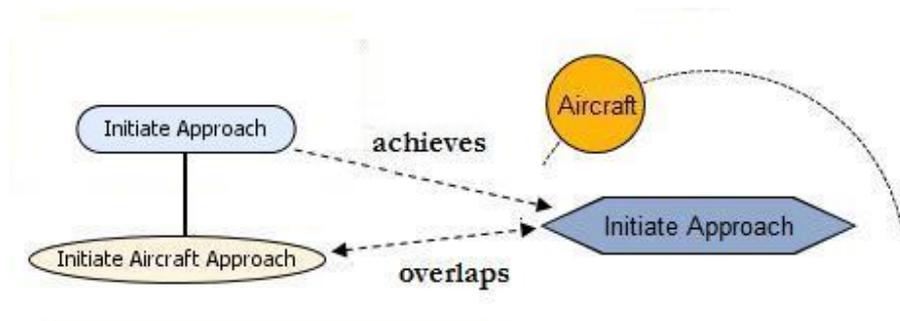


Figure F.38 Prometheus Plan vs SR Task achieves traceability relation

- Prometheus Percept vs SD Resource – a percept p_1 in Prometheus has an *overlaps* traceability relation with a SD resource r_1 in i^* when the name of percept p_1 is synonyms to the name to the SD Resource r_1 . For instance, the name of the *Credit Card Details* SD Resource is synonyms to the name of *Credit Card Details* percept in Prometheus (see Figure F.39). Therefore, there is an *overlaps* traceability relation between *Credit Card Details* SD Resource and *Credit Card Details* percept.

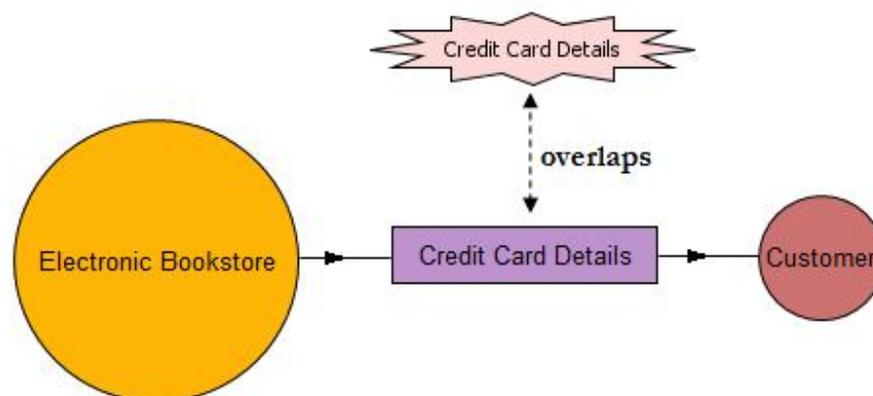


Figure F.39 Prometheus Percept vs SD Resource overlaps traceability relation

- Prometheus Action vs SR Task – an action a_1 in Prometheus has an *overlaps* traceability relation with a SR Task t_1 in i^* when the name of the action a_1 is synonyms to the

name to the SR Task t_i . For instance, the name of *Place Delivery Request* action is synonyms to the name of the *Place Delivery Request* SR task (see Figure F.40). Therefore, there is an *overlaps* traceability relation between *Place Delivery Request* action and *Place Delivery Request* SR Task.

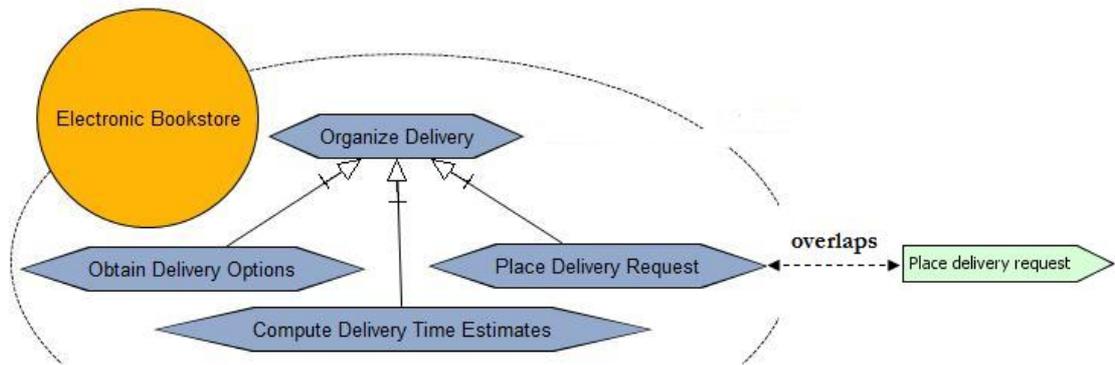


Figure F.40 Prometheus Action vs SR Task overlaps traceability relation

- Prometheus Data vs SD Goal – a data d_1 in Prometheus has a *contributes* traceability relation with a SD Goal g_1 in i^* when some of the sub-resources of the SD Goal g_1 has an *overlaps* traceability relation with the data d_1 . For instance, Customer Order SR resource is sub-resource of the Buy Book SD goal and Customer Order SR Resource has an overlaps traceability relation with Customer Order Prometheus data (see Figure F.41). Therefore, there is a contributes traceability relation between Customer Order data in Prometheus and Buy Book SD goal in i^* .

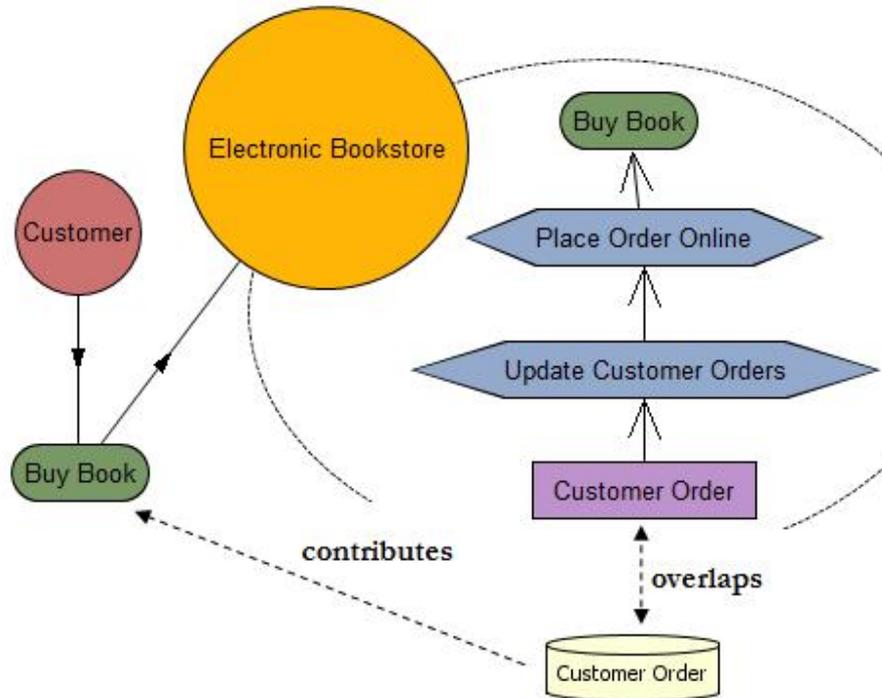


Figure F.41 Prometheus Data vs SD Goal contributes traceability relation

- Prometheus Data vs SD Task - a data d_1 in Prometheus has a *contributes* traceability relation with a SD Task t_1 in i^* when some of the sub-resources of the SD Goal g_1 has an *overlaps* traceability relation with the data d_1 . For instance, balances SD resource in i^* is an sub-resource of the Process Withdraw SD task and balances SD resource has an overlaps traceability relation with balances data in Prometheus (see Figure F.42). Therefore, there is a contributes traceability relation between balances data in Prometheus and Process Withdraw SD task in i^* .

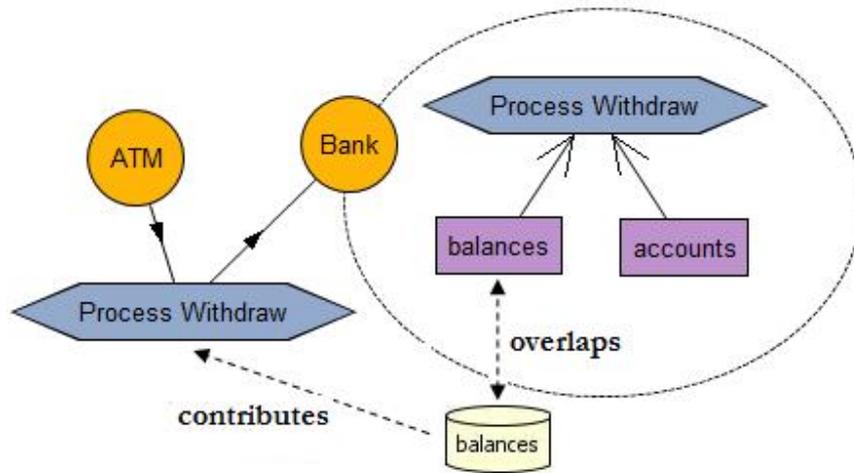


Figure F.42 Prometheus Data vs SD Task contributes traceability relation

- Prometheus Data vs Actor – a data d_1 in Prometheus has a *uses* traceability relation with an actor a_1 when the actor a_1 has a SR Resource r_1 that has an *overlaps* traceability relation with the data d_1 . For instance, Electronic Bookstore actor in i^* has Customer Order SR resource and Customer Order SR resource has an *overlaps* traceability relation with Customer Order data in Prometheus (see Figure F.43). Therefore, there is an *uses* traceability relation between Electronic Bookstore in i^* and Customer Order data in Prometheus.

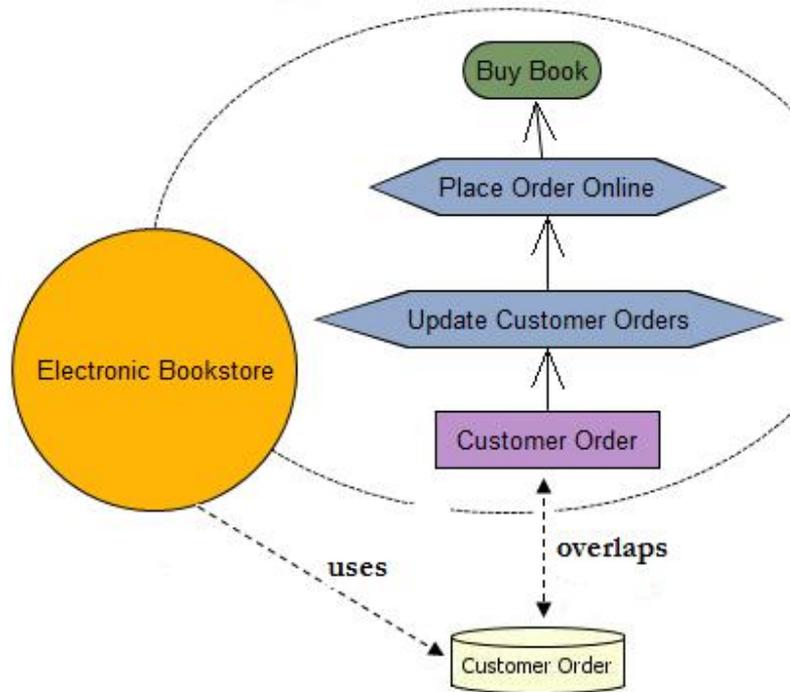


Figure F.43 Prometheus Data vs Actor uses traceability relation

- Prometheus Data vs SR Goal - a data d_1 in Prometheus has an *uses* traceability relation with a SR Goal g_1 in i^* when some of the sub-resources of the SR Goal g_1 has an *overlaps* traceability relation with the data d_1 . For instance, Customer Order SR resource in i^* is a sub-resource of Buy Book SR goal and Customer Order SR resource has an *overlaps* traceability relation with Customer Order data in Prometheus (see Figure F.44). Therefore, there is an *uses* traceability relation between Buy Book SR goal and Customer Order data in Prometheus.

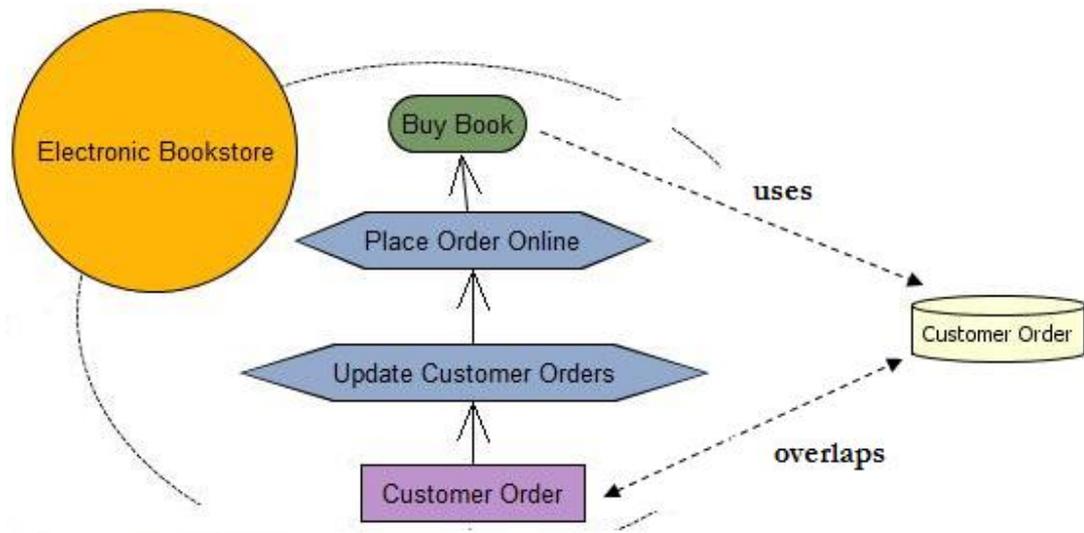


Figure F.44 Prometheus Data vs SR Goal uses traceability relation

- Prometheus Data vs SR Resource – a data d_1 in Prometheus has an *overlaps* traceability relation with a SR Resource r_1 in i^* when the name of the data d_1 is synonyms to the name of the SR Resource. For instance, Customer Order SR resource has a synonyms name to the of the Customer Order data in Prometheus (see Figure F.45) Therefore, there is an overlaps traceability relation between Customer Order SR resource and Customer Order data in Prometheus.

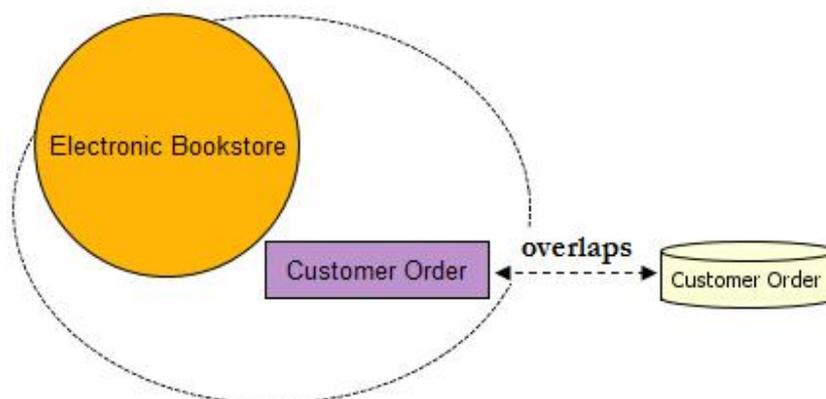


Figure F.45 Prometheus Data vs SR Resource overlaps traceability relation

- Prometheus Data vs SR Task – a data d_1 in Prometheus has an *uses* traceability relation with a SR Task t_1 in i^* when some of the sub-resources of the SR Task t_1 has an *overlaps* traceability relation with the data d_1 . For instance, balances SR resource in i^* is a sub-

resource of the Process Withdraw and balances SR resource has an overlaps traceability relation with balances data in Prometheus (see Figure F.46). Therefore, there is an uses traceability relation between Process Withdraw SR task and balances data in Prometheus.

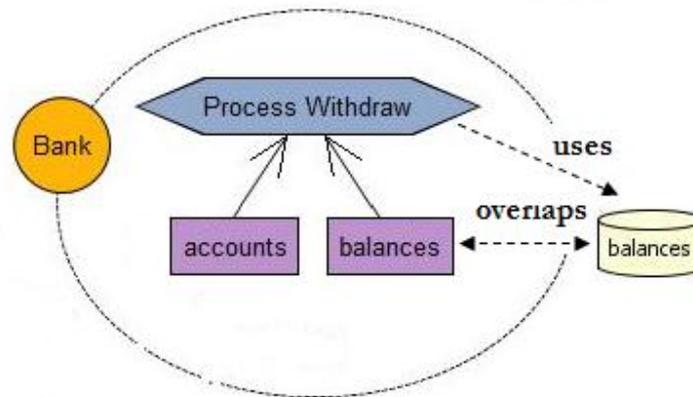


Figure F.46 Prometheus Data vs SR Task uses traceability relation

- Prometheus Scenario vs SD Goal – a scenario s_1 in Prometheus has a *depends on* traceability relation with a SD Goal g_1 in i^* when the number of sub-elements of the goal g_1 that has an *overlaps* traceability relation with the steps of the scenario s_1 is greater than a threshold (e.g. 80%) and the name of the scenario is synonyms to the name of the SD Goal. For instance, Order book scenario (see Figure F.47) is composed of the following steps: Get delivery options goal, Calculate delivery time estimates goal, Present information goal, Delivery Choice percept, Get credit card details goal, Credit Card Details percept, Execute bank transaction action, Place delivery request action, Log delivery problems goal, Update customers orders goal, Send book order action . Buy Book SD goal (see Figure F.48) is decomposed in Place Order Online SR task, Place Order By Phone SR task, Send Book Order Confirmation SR task, Update Customer Orders SR task, Customer Order SR task, Make Payment SR task, Perform Bank Transaction SR task, Transaction Accepted SR task, Transaction Rejected SR task, Obtain Credit Card Details SR task, Delivery Handling SR task, Fill Pending Order SR task, Organize Delivery SR task, Log Outgoing Delivery SR task, Place Delivery Request SR task, Compute Delivery Time Estimates SR task, and Obtain Delivery Options SR task, Postal DB SR resource, Courier DB SR resource. Get

delivery option goal in Prometheus has an overlaps relation with Obtain Delivery Options SR task, Calculate delivery time has an overlaps traceability relation with estimates goal and Compute Delivery Time Estimates SR task, Delivery choice percept has an overlaps traceability relation with Delivery Choice SD resource, Get credit card details goal has an overlaps traceability with Obtain Credit Card Details SR task, Credit Card Details percept has an overlaps traceability Credit Card Details SD resource, Execute bank transaction action has an overlaps traceability relation with Perform Bank Transaction SR task, Place delivery request action has an overlaps traceability relation with Place Delivery Request SR task, Log delivery problems goal has an overlaps traceability relation with Log Delivery Problems SR task, Update custome orders goal has an overlaps traceability relation with Update Customer Orders SR task, Send book order action has an overlaps traceability relation with Send Book Order Confirmation SR task. Therefore, there is a *depends* traceability relation between Order book scenario and Buy Book SD goal since 90,90% of steps of the Order book scenario has an overlaps traceability relation with sub-elements of the Buy Book SD goal and Order book and Buy Book are synnomys .

	Type	Name	Role	...	Data
1	G	Get delivery options	Delivery Handling		
2	G	Calculate delivery time e...	Delivery Handling		
3	G	Present information	Online Interaction		
4	P	Delivery Choice	Delivery Handling		
5	G	Get credit card details	Purchasing		
6	P	Credit Card Details	Purchasing		
7	A	Execute bank transaction	Purchasing		
8	A	Place delivery request	Delivery Handling		
9	G	Log delivery problems	Stock Management		
10	G	Update customers orders	Purchasing		Customer Order
11	A	Send book order	Purchasing		

A -> Action G -> Goal O -> Others P -> Percept S -> Scenario

Figure F.47 Order Book Scenario

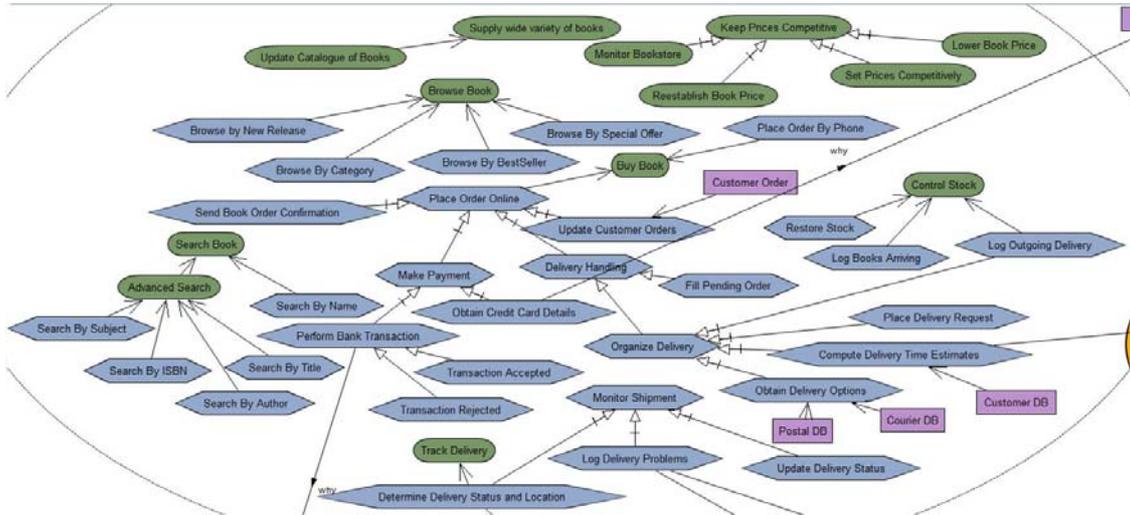


Figure F.48 Strategic Rationale Diagram for the Electronic Bookstore actor

- Prometheus Scenario vs SD Resource – a scenario s_1 in Prometheus has a *composed of* traceability relation with a SD Resource r_1 when one of the steps of the scenario s_1 has an *overlaps* traceability with the SD Resource r_1 . For instance, the *Order book* scenario (see Figure F.47) has the *Credit Card Details* step that has an *overlaps* traceability relation with the *Credit Card Details* SD Resource in i^* . Therefore, there is a composed of traceability relation between *Order book scenario* and *Credit Card Details*.
- Prometheus Scenario vs SD Task – a scenario s_1 in Prometheus has a *depends on* traceability relation with a SD Task t_1 in i^* when the number of sub-elements of the goal t_1 that has an *overlaps* traceability relation with the steps of the scenario s_1 is greater than a threshold (e.g. 80%) and the name of the scenario is synonyms to the name of the SD task. For instance, Monitor delivery scenario (see Figure F.49) is composed of Determine delivery status goal, Log delivery problems goal and Update delivery status goal. Monitor Shipment SD task (see Figure F.48) is decomposed on Determine Delivery Status and Location SR task, Log Delivery Problems SR task, and Update Delivery Status SR task. Determine delivery status has an overlaps traceability relation with Determine Delivery Status and Location SR task, Log delivery problems goal has an overlaps traceability relation with Log Delivery Problems SR task and Update delivery status goal has an overlaps traceability relation with Update Delivery Status SR task. Therefore, there is a *depends* traceability relation between Monitor delivery

scenario and Monitor Shipment SD goal since 100% of steps of the Order book scenario has an overlaps traceability relation with sub-elements of the Monitor Shipment SD goal and Monitor Shipment and Monitor delivery are synonyms .

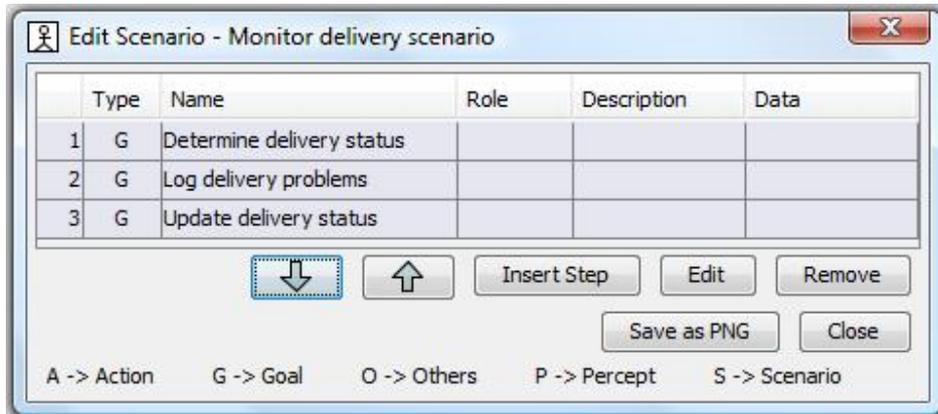


Figure F.49 Prometheus Scenario vs SD Task depends traceability relation

- Prometheus Scenario vs Actor – a scenario s_1 has a *depends on* traceability relation with an actor a_1 when there is an agent a_2 in Prometheus realises the scenario s_1 and it has an *overlaps* traceability relation with the actor a_1 . For instance Aircraft agent in Prometheus has an overlaps traceability relation with Aircraft actor in i^* and Aircraft agent realises Landing scenario (see Figure F.50). Therefore, there is a depends traceability relation between Landing scenario in Prometheus and Aircraft actor in i^* .

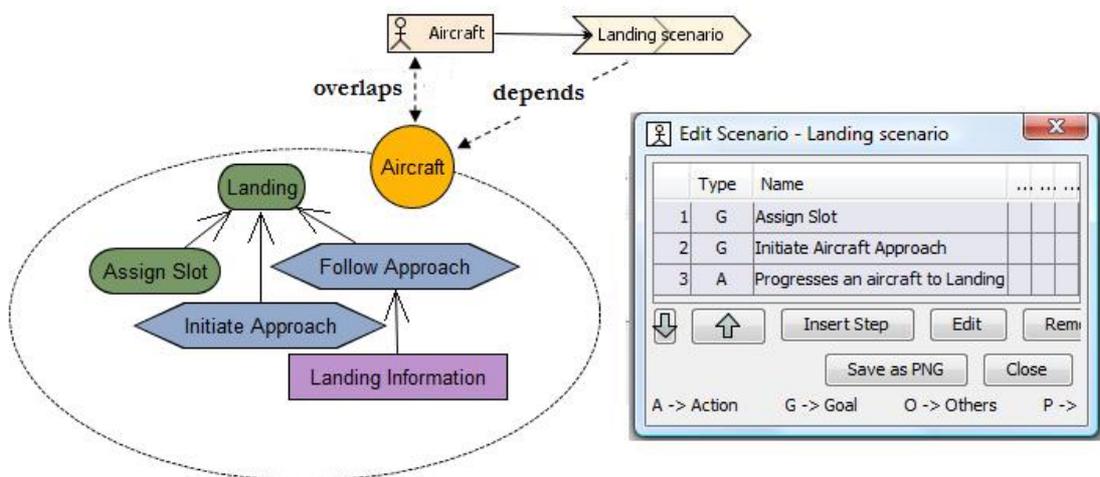


Figure F.50 Prometheus Scenario vs Actor depends traceability relation

- Prometheus Scenario vs SR Goal – a scenario s_1 in Prometheus has a *composed* traceability relation with a SR Goal g_1 in i^* when a step of the scenario s_1 and the goal g_1 has an *overlaps* traceability relation. For instance, Assign Slot SR goal has an *overlaps* traceability with Assign Slot step of the Landing scenario (see Figure F.51). Therefore, there is a composed traceability relation between Landing scenario and Assign Slot SR goal.

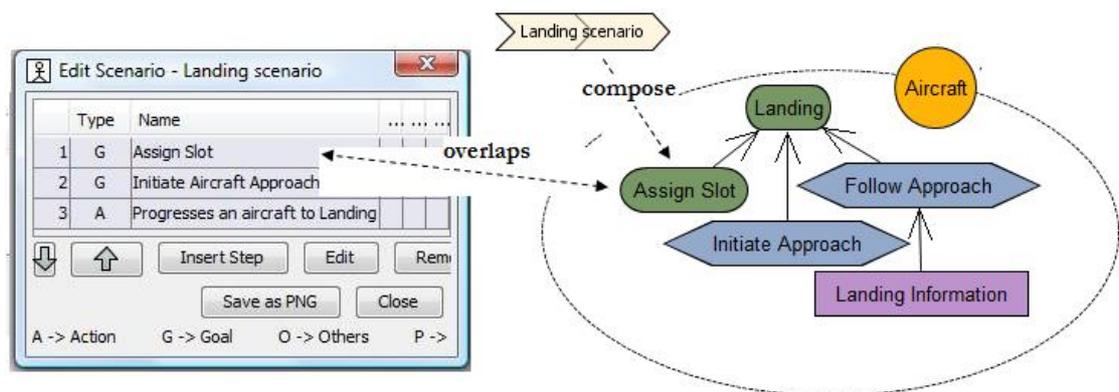


Figure F.51 Prometheus Scenario vs SR Goal compose traceability relation

- Prometheus Scenario vs SR Resource – a scenario s_1 in Prometheus has a *creates* traceability relation with a SR resource r_1 in i^* when one of the steps of the scenario s_1 writes on data that has an *overlaps* traceability relation with resource r_1 . For instance, Update customer orders step writes on Customer Order data that has an *overlaps* traceability relation with Customer Order SR resource (see Figure F.52). Therefore, there is a *creates* traceability relation between Order book scenario and Customer Order SR resource.

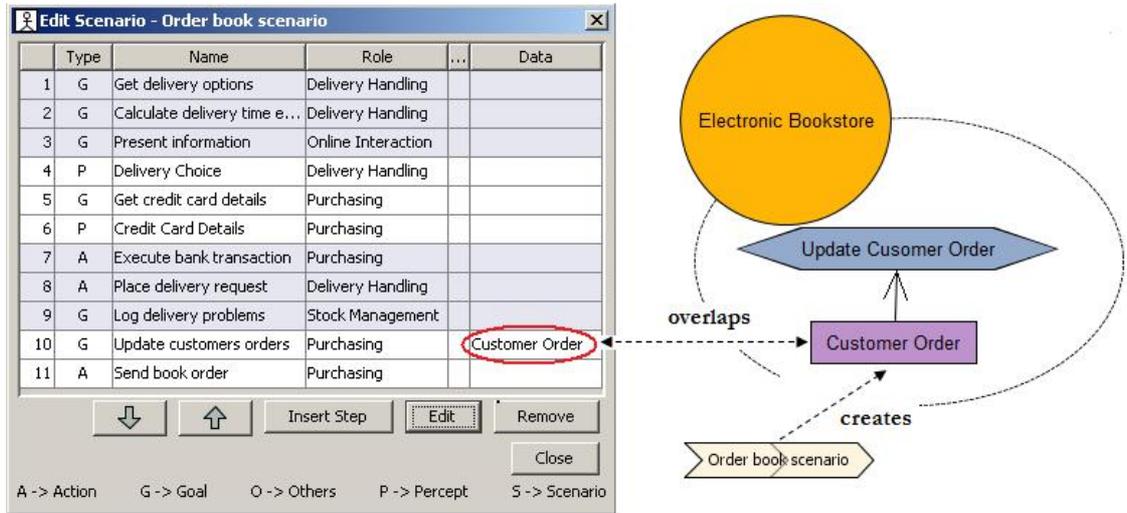


Figure F.52 Prometheus Scenario vs SR Resource creates traceability relation

- Prometheus Scenario vs SR Resource – a scenario s_1 in Prometheus has an uses traceability relation with a SR resource r_1 in i^* when one of the steps of the scenario s_1 reads a data that has an *overlaps* traceability relation with resource r_1 . For instance, Progresses an aircraft to Landing step reads landing_info data that has overlaps traceability relation with Landing Information SR resource (see Figure F.53). Therefore, there is a uses traceability relation between Landing scenario and Landing Information SR resource.

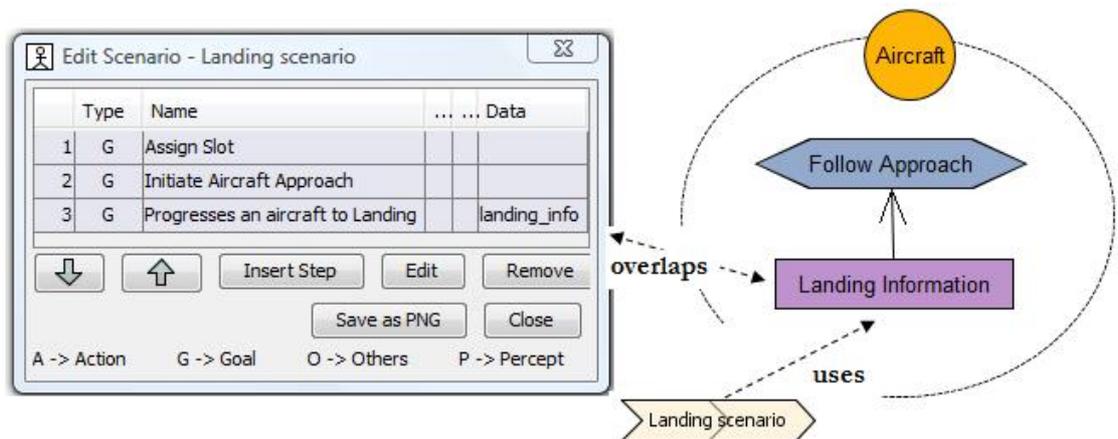


Figure F.53 Prometheus Scenario vs SR Resource uses traceability relation

- Prometheus Scenario vs SR Task - a scenario s_1 in Prometheus has a *composed* traceability relation with a SR Task t_1 in i^* when a step of the scenario s_1 and the task t_1 has an overlaps traceability relation. For instance, Initiate Approach SR task has an overlaps traceability with Initiate Aircraft Approach step of the Landing scenario (see Figure F.54). Therefore, there is a composed traceability relation between Landing scenario and Initiate Aircraft SR task.

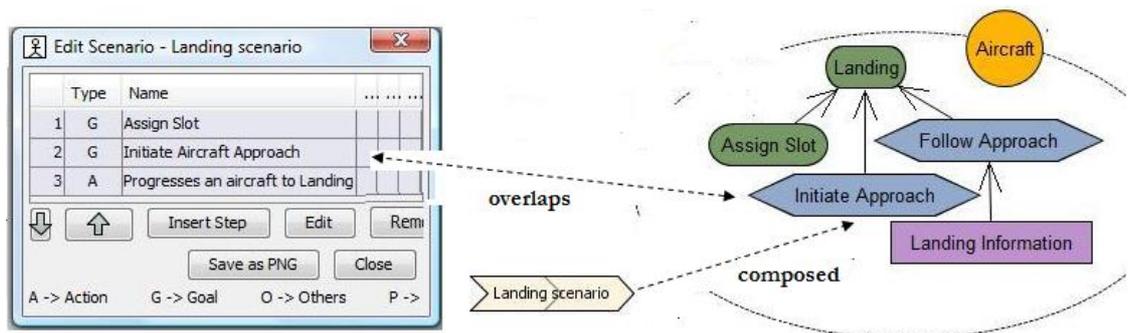


Figure F.54 Prometheus Scenario vs SR Task composed traceability relation

- Prometheus Message vs SD Resource – a message m_1 in Prometheus has an *overlaps* traceability relation with a SD resource r_1 when the message m_1 has an carried information that is synonyms to the name of the SD resource r_1 . For instance, Aircraft Event message has carry Slot Allocated information that is synonyms to Slot Allocated SR resource (see Figure F.55). Therefore, there is an overlaps traceability relation between AircraftEvent message in Prometheus and Slot Allocated SD resource r_1 .

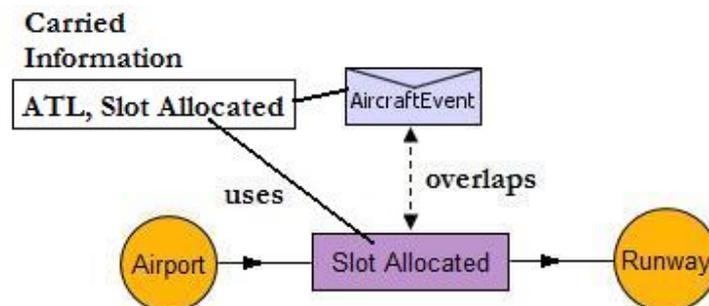


Figure F.55 Prometheus Message vs SD Resource overlaps traceability relation

Appendix G - Traceability Relations between Prometheus and JACK

This appendix describes traceability relations between Prometheus and JACK elements.

We have identified seven different types of traceability relations between the various elements in the models used in our approach. The types of traceability relations are *overlaps*, *contributes to*, *uses*, *creates*, *achieves*, *depends on*, and *composed of*. We present below descriptions of these different types of relations.

- **Overlaps** – in this type of relation, an element e1 overlaps with an element e2 (an element e2 overlaps with an element e1), if e1 and e2 refer to elements with common aspects of the agent software development. As shown in Tables 3.3 and 3.4, an *overlaps* relation may hold between a) an agent in JACK and an agent in Prometheus; b) a plan in JACK and a plan in Prometheus; c) a beliefSet in JACK and a data in Prometheus; an event in JACK and Prometheus message.
- **Uses (Used by)** - in this type of relation, an element e1 uses an element e2, if e1 requires the existence of e2 in order to achieve its objective. As shown in Tables 3.3 and 3.4, a *contributes* relation may hold between a) an agent in JACK and a role in Prometheus; b) a plan in JACK and a role in Prometheus; c) a plan in JACK and a capability in Prometheus; d) a beliefSet in JACK and a role in Prometheus; e) a beliefSet in JACK and a capability in Prometheus;
- **Creates (Created by)** - in this type of relation an element e1 creates an element e2, if e1 generates element e2. As shown in Tables 3.3 and 3.4, a *creates* relation may hold between a) a plan in Prometheus and an actor in *i**; b) role in Prometheus and SR resource in *i**; c) an agent in Prometheus and a SR resource in *i**; d) a capability in Prometheus and a SR resource in *i**; e) a plan in Prometheus and a SR resource in *i**; f) a scenario in Prometheus and a SR resource in *i**.

- Achieves (Achieved by) - in this type of relation an element e1 achieves an element e2, if e1 meets the expectations and needs of e2. As shown in Tables 3.3 and 3.4, a *achieves* relation may hold between a) an agent in JACK and a goal in Prometheus; b) a plan in JACK and a goal in Prometheus.
- Sends (Is Sent by) – in this type of relation an element e1 sends an element e2, if e1 is responsible to create an element and send this element to another element e2. As shown in Tables 3.3 and 3.4, a sends relation may hold between a) an event in JACK and an agent in Prometheus; b) an event in JACK and a capability in Prometheus; c) an agent in JACK and a message in Prometheus.
- Receives (Is Received by) – in this type of relation an element e1 receives an element e2, if e1 receives an element created by another element e2. As shown in Tables 3.3 and 3.4, a receives relation may hold between a) an event in JACK and an agent in Prometheus; b) an event in JACK and a capability in Prometheus; c) an agent in JACK and a message in Prometheus; d) an event in JACK and a plan in Prometheus.

Tables G.1 and G.2 present different types of traceability relations for the main types of elements in Prometheus and JACK models. Tables G.1 and G.2, apart from overlaps relations that are bi-directional, the direction of a relation is represented from a row[i] to a column[j] (e.g. “An agent in JACK achieves a goal in Prometheus”).

Prometheus JACK	Goal	Role	Agent	Capability
Agent	Achieves	Uses	Overlaps	Uses
Plan	Achieves	Used by	Is Used by	Used by
BeliefSet	---	Creates/Uses	Creates/Uses	Creates/Uses
Event	---	---	Is Sent by/ Is Received by	Is Sent by/ Is Received by

Table G.1 Traceability Relations Types between Prometheus and JACK Artefacts

Prometheus JACK	Plan	Percept	Action	Message	Data
Agent	Uses	Uses	Creates	Send/Receives	Uses/Creates
Plan	Overlaps	Uses	Creates	Send/Receives	Uses/Creates
BeliefSet	Creates/Uses	---	---	---	Overlaps
Event	Send/Receives	---	---	Overlaps	---

Table G.2 Traceability Relations Types between Prometheus and JACK Artefacts

- JACK Agent vs Prometheus Goal – an agent a_1 in JACK has an *achieves* traceability relation with a goal g_1 in Prometheus when there is an *overlaps* traceability relation between the JACK agent a_1 and an agent in Prometheus a_2 and the goal g_1 is one of the goals that the Prometheus agent a_2 achieves. For instance, Sales Assistant agent in Prometheus has an *overlaps* traceability relation with the SalesAssistant agent in JACK and the Sales Assistant agent in Prometheus *achieves* the Respond Add Customer Request goal in Prometheus (see Figure G.1). Therefore, there is an *achieves* relation between the SalesAssistant agent in JACK and the Respond Add Customer Request goal in Prometheus.

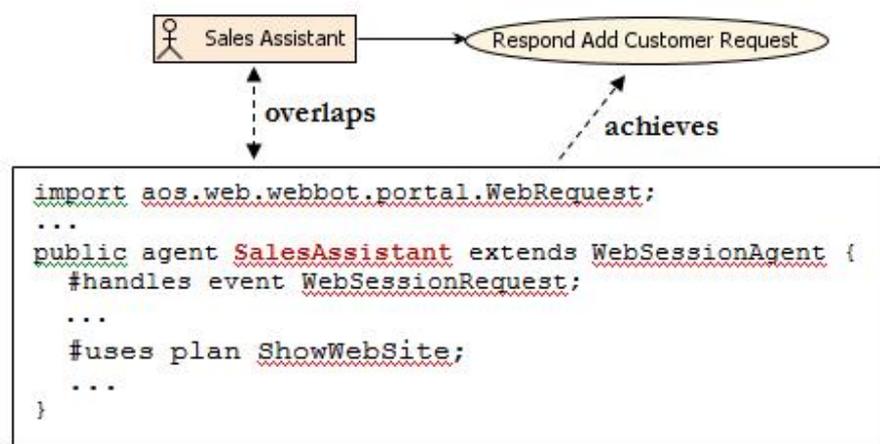


Figure G.1 JACK Agent vs Prometheus Goal achieves traceability relation

- JACK Agent vs Prometheus Role - an agent a_1 in JACK has a *uses* traceability relation with a role r_1 in Prometheus when there is an *overlaps* traceability relation between JACK agent a_1 and an agent in Prometheus a_2 and the role r_1 is one of the roles that the Prometheus agent a_2 includes. For instance, Sales Assistant agent in Prometheus has an *overlaps* traceability relation with the SalesAssistant agent in JACK and the Sales Assistant agent in Prometheus includes Order Management role (see Figure G.2). Therefore, there is an *uses* traceability relation between SalesAssistant agent in JACK and Order Management role in Prometheus.

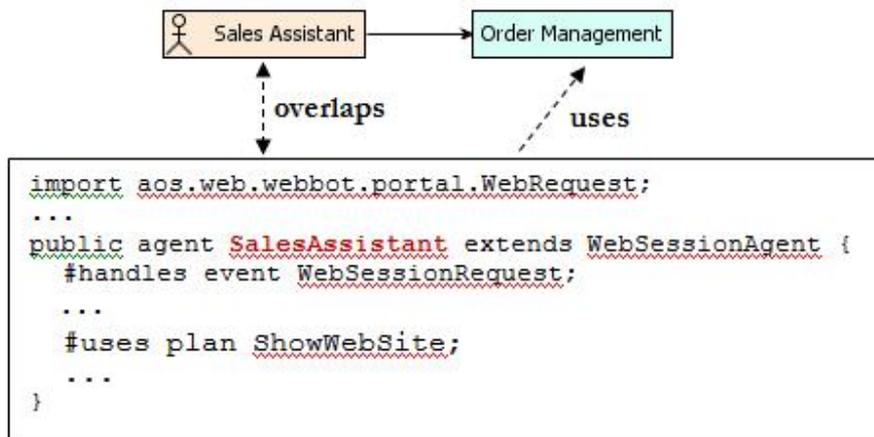


Figure G.2 JACK Agent vs Prometheus Role uses traceability relation

- JACK Agent vs Prometheus Agent - an agent a_1 in JACK have an *overlaps* traceability relation with an agent a_2 in Prometheus when the name of the agent a_1 in JACK is synonyms to the name of the agent a_2 in Prometheus. For instance, Sales Assistant agent in Prometheus has *synonyms* name to SalesAssistant agent in JACK (see Figure G.3). Therefore there is an overlaps traceability relation between Sales Assistant agent in Prometheus and SalesAssistant agent in JACK.

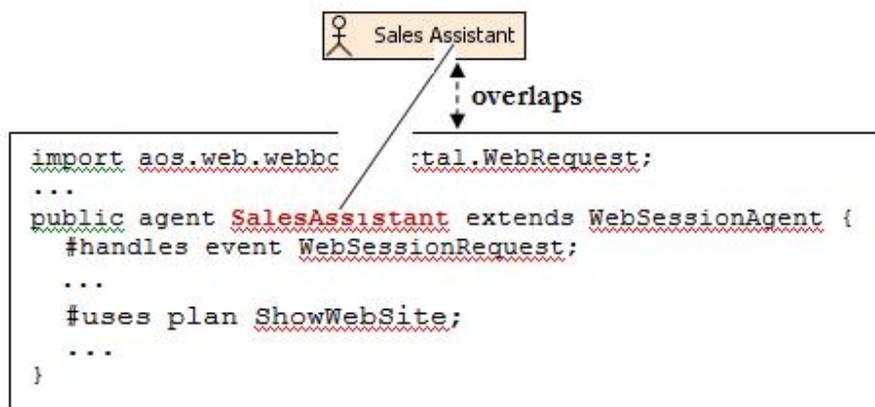


Figure G.3 JACK Agent vs Prometheus Agent overlaps traceability relation

- JACK Agent vs Prometheus Capability – an agent in JACK has *uses* traceability relation when there is *overlap* traceability relation between a JACK agent and a Prometheus agent and the Prometheus agent has the Prometheus capability. For instance, Runway agent in Prometheus has an overlaps traceability relation with Runway agent in JACK and Runway agent in Prometheus has ArrivalSequencing capability (see Figure G.4).

Therefore, there is an uses traceability relation between Runway agent in JACK and ArrivalSequencing capability in Prometheus.

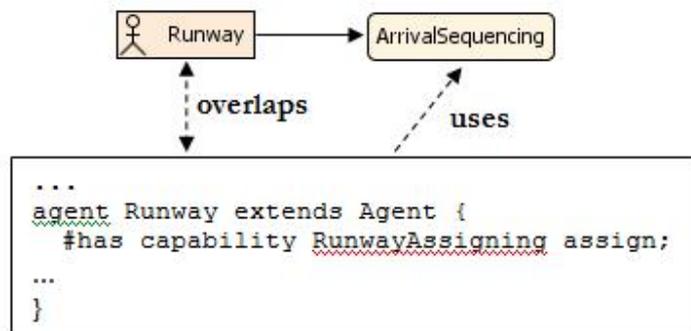


Figure G.4 JACK Agent vs Prometheus Capability uses traceability relation

- JACK Agent vs Prometheus Plan – an agent a_1 in JACK has *uses* traceability relation with a plan p_1 in Prometheus when there is *overlap* traceability relation between the JACK agent a_1 and a Prometheus agent a_2 in Prometheus and the agent a_2 includes the plan p_1 in Prometheus. For instance, BankAgent agent in JACK has an overlaps traceability relation with Bank agent in Prometheus and Bank agent in Prometheus uses Process Withdraw plan (see Figure G.5). Therefore, there is an uses traceability relation between BankAgent in JACK and Process Withdraw plan in Prometheus.

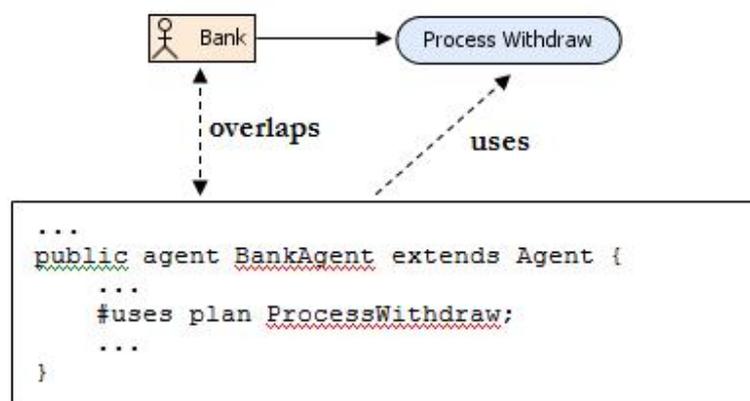


Figure G.5 JACK Agent vs Prometheus Plan uses traceability relation

- JACK Agent vs Prometheus Percept – an agent a_1 in JACK has *uses* traceability relation with a percept p_1 in Prometheus when there is an *overlaps* traceability relation between the agent a_1 in JACK and an agent a_2 in Prometheus and the Prometheus agent a_2 in

Prometheus responds to the percept p_1 . For instance, Bank agent in Prometheus has an overlaps traceability relation with BankAgent agent in JACK and Bank agent responds Withdraw Percept percept (see Figure G.6). Therefore, there is an uses traceability relation between BankAgent in JACK and Withdraw Percept in Prometheus.

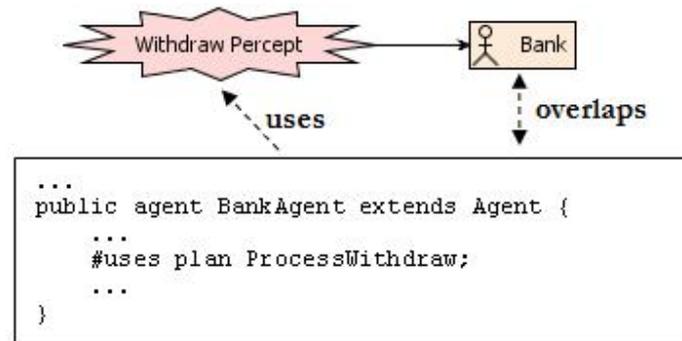


Figure G.6 JACK Agent vs Prometheus Percept uses traceability relation

- JACK Agent vs Prometheus Action - an agent ag_1 in JACK has *creates* traceability relation with an action a_1 in Prometheus when there is an *overlaps* traceability relation between the agent ag_1 in JACK and an agent ag_2 in Prometheus and the Prometheus agent ag_2 in Prometheus includes the action a_1 . For instance, Bank agent in Prometheus has an overlaps traceability relation with BankAgent in JACK and BankAgent performs Dispense Cash action (see Figure G.7). Therefore, there is a creates traceability relation between BankAgent agent in JACK and Dispense Cash.

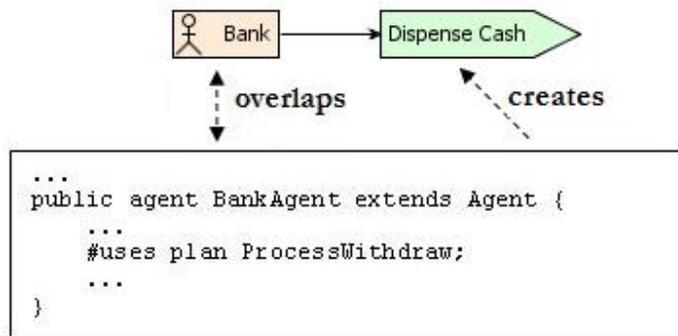


Figure G.7 JACK Agent vs Prometheus Action creates traceability relation

- JACK Agent and Prometheus Message (*sends*) – an agent ag_1 in JACK has *sends* traceability relation with a message m_1 in Prometheus when there is an *overlaps* traceability relation between the agent ag_1 in JACK and an agent ag_2 in Prometheus and the Prometheus agent ag_2 in Prometheus *sends* the message m_1 . For instance, Bank agent in Prometheus has an overlaps traceability relation with BankAgent in JACK and Bank agent in Prometheus sends Withdraw Response message (see Figure G.8). Therefore, there is a sends traceability relation between BankAgent in JACK and Withdraw Response message in Prometheus.

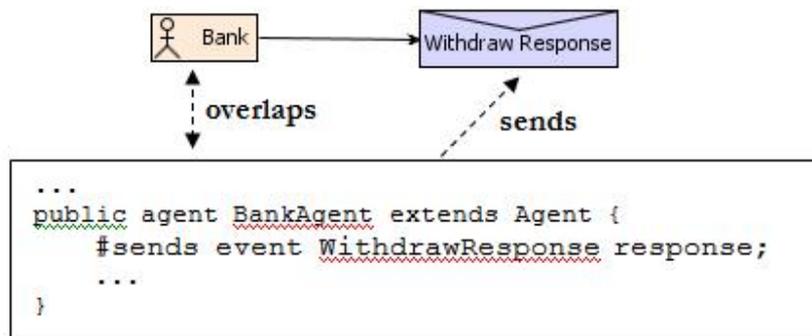


Figure G.8 JACK Agent vs Prometheus Message sends traceability relation

- JACK Agent vs Prometheus Message (*receives*) – an agent ag_1 in JACK has *receives* traceability relation with a message m_1 in Prometheus when there is an *overlaps* traceability relation between the agent ag_1 in JACK and an agent ag_2 in Prometheus and the Prometheus agent ag_2 in Prometheus *receives* the message m_1 . For instance, Bank agent in Prometheus has an overlaps traceability relation with BankAgent in JACK and Bank agent in Prometheus receives Withdraw Request message (see Figure

G.9). Therefore, there is a receives traceability relation between BankAgent in JACK and Withdraw Request message in Prometheus.

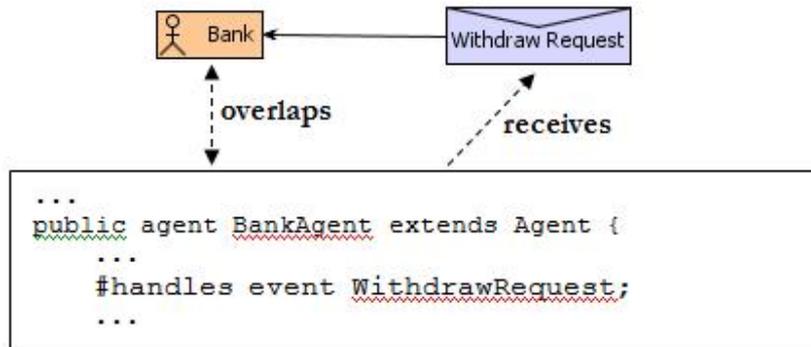


Figure G.9 JACK Agent vs Prometheus Message receives traceability relation

- JACK Agent and Prometheus Data (*uses*) - an agent ag_1 in JACK has *uses* traceability relation with a data d_1 in Prometheus when there is an *overlaps* traceability relation between the agent ag_1 in JACK and an agent ag_2 in Prometheus and the Prometheus agent ag_2 *uses* the data d_1 . For instance, Bank agent in Prometheus has an overlaps traceability relation with BankAgent in JACK and Bank agent in Prometheus has balances data (see Figure G.10). Therefore, there is an uses traceability relation between BankAgent in JACK and balances data in Prometheus.

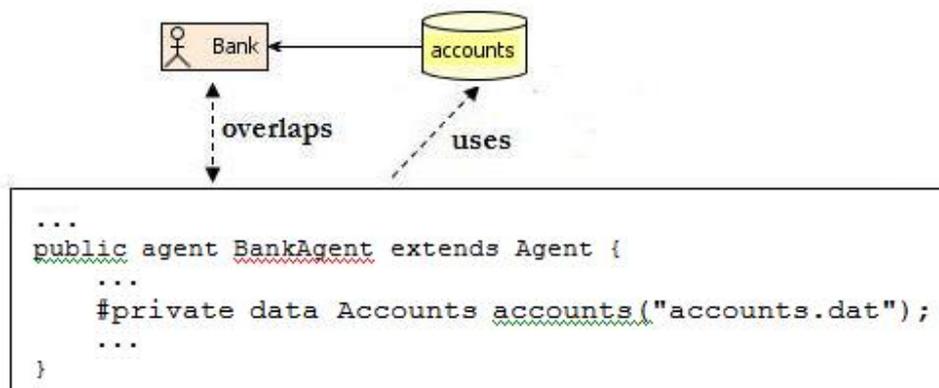


Figure G.10 JACK Agent vs Prometheus Message receives traceability relation

- JACK Agent vs Prometheus Data (*creates*) - an agent ag_1 in JACK has *creates* traceability relation with a data d_1 in Prometheus when there is an *overlaps* traceability relation between the agent ag_1 in JACK and an agent ag_2 in Prometheus and the Prometheus

agent ag_2 creates the data d_1 . For instance, Bank agent in Prometheus has an overlaps traceability relation with BankAgent in JACK and Bank agent in Prometheus creates balances data in Prometheus (see Figure G.11). Therefore, there is a creates traceability relation between BankAgent in JACK and balances data in Prometheus.

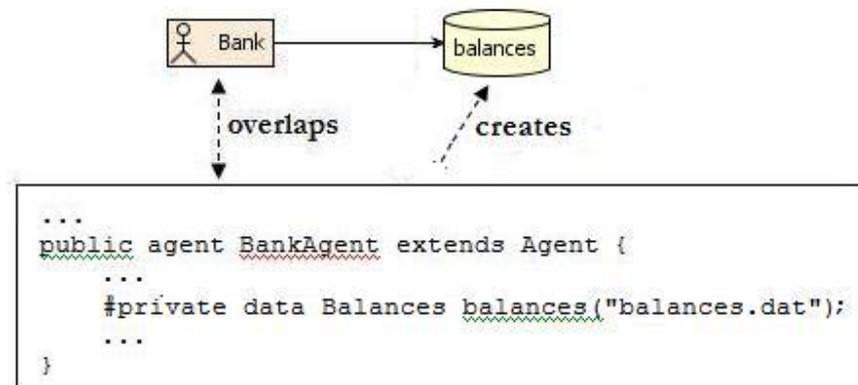


Figure G.11 JACK Agent vs Prometheus Date creates traceability relation

- JACK Plan vs Prometheus Goal – a plan p_1 in JACK has *achieves* traceability relation with a goal g_1 in Prometheus when there is an *overlaps* traceability relation between the plan p_1 in JACK and the plan p_2 in Prometheus and the Prometheus plan p_2 *achieves* the goal g_1 . For instance, Withdraw Cash plan in Prometheus has an overlaps traceability relation with WithdrawCash plan in JACK and Withdraw Cash plan achieves Withdraw Money goal in Prometheus (see Figure G.12). Therefore, there is an achieves traceability relation between WithdrawCash plan in JACK and Withdraw Money goal in Prometheus.

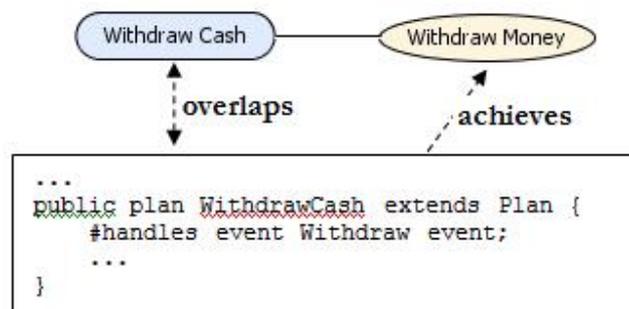


Figure G.12 JACK Plan vs Prometheus Goal

- JACK Plan vs Prometheus Role – a plan p_1 in JACK has *uses* traceability relation with a role r_1 in Prometheus when there is an *overlaps* traceability relation between the plan p_1 in JACK and a plan p_2 in Prometheus and the Prometheus role includes the plan p_2 (see Figure G.13). For instance, Book Finding role in Prometheus includes Find Best Sellers plan that has an overlaps traceability relation with FindBestSellers plan in JACK. Therefore, there is an uses traceability relation between Book Finding role in Prometheus and FindBestSellers plan in JACK.

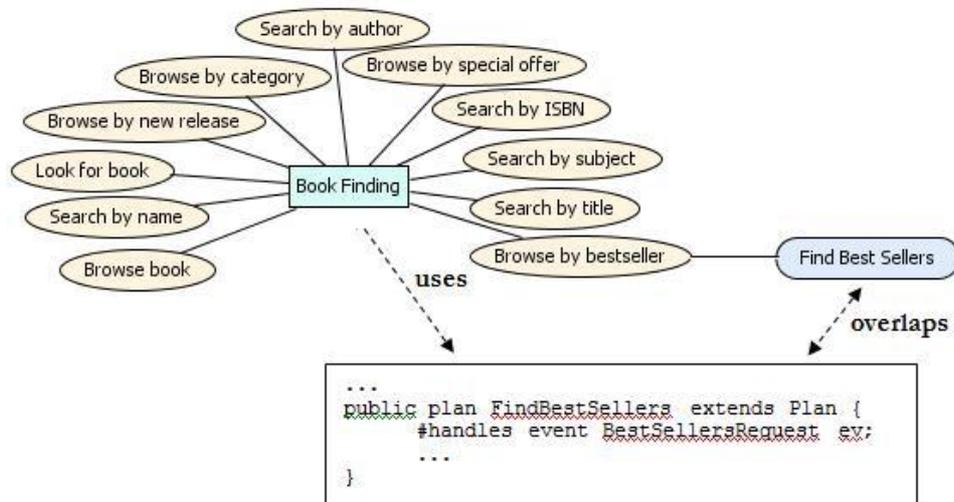


Figure G.13 JACK Plan vs Prometheus Role uses traceability relation

- JACK Plan vs Prometheus Agent - a plan p_1 in JACK has *uses* traceability relation with an agent a_1 in Prometheus when there is an *overlaps* traceability relation between the plan p_1 in JACK and a plan p_2 in Prometheus and the Prometheus agent a_1 includes the plan p_2 . For instance, Atm agent in Prometheus uses Withdraw Cash plan that has an overlaps traceability relation with WithdrawCash plan in JACK (see Figure G.14). Therefore, there is an uses traceability relation between Atm agent in Prometheus and WithdrawCash plan in JACK.

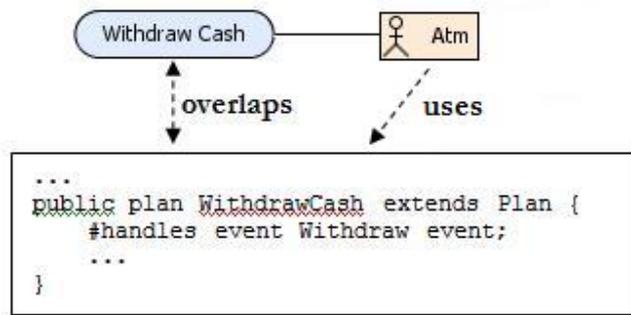


Figure G.14 JACK Plan vs Prometheus Agent uses traceability relation

- JACK Plan vs Prometheus Capability - a plan p_1 in JACK has *uses* traceability relation with a capability c_1 in Prometheus when there is an *overlaps* traceability relation between the plan p_1 in JACK and a plan p_2 in Prometheus and the Prometheus capability uses the plan p_2 . For instance, ArrivalSequencing capability has Request Slot Plan plan and Request Slot Plan plan has an overlaps traceability relation with RequestSlot plan in JACK (see Figure G.15). Therefore, there is an uses traceability relation between ArrivalSequencing capability and Request Slot plan in JACK.

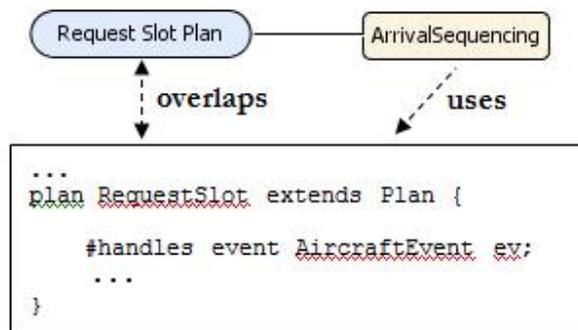


Figure G.15 JACK Plan vs Prometheus Capability uses traceability relation

- JACK Plan vs Prometheus Plan – a plan in Prometheus and a plan in JACK have an *overlaps* traceability relation when the name of the plan in JACK is *synonyms* to the name of the plan in Prometheus and name of the message that triggers the plan has to be *synonyms* to the name of the event. The name of the message that triggers the plan has to be *synonyms* to the name of the event. For instance, Request Slot Plan in Prometheus has synonyms name to RequestSlot plan in JACK (see Figure G.16). Therefore, there is an overlaps traceability relation between Request Slot Plan plan and RequestSlot plan in JACK.

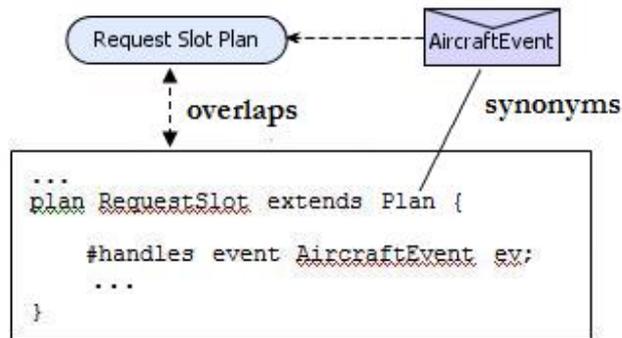


Figure G.16 JACK Plan vs Prometheus Plan overlaps traceability relation

- JACK Plan vs Prometheus Percept - a plan p_1 in JACK has an uses traceability relation with a percept p_1 when there is a plan p_2 in Prometheus that has an overlaps traceability relation with the plan p_1 and the plan p_2 responds to the percept p_1 . For instance, Withdraw Cash plan in Prometheus responds to the Withdraw Percept percept and WithdrawCash plan in JACK has an overlaps traceability relation with Withdraw Cash plan in Prometheus (see Figure G.17). Therefore, there is an uses traceability relation between WithdrawCash plan in JACK and Withdraw Percept percept in Prometheus.

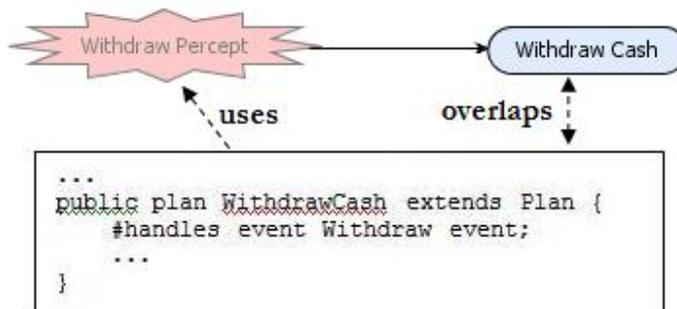


Figure G.17 JACK Plan vs Prometheus Percept uses traceability relation

- JACK Plan vs Prometheus Action (*creates*)- a plan p_1 in JACK has *creates* traceability relation with an action a_1 in Prometheus when there is an *overlaps* traceability relation between the plan p_1 in JACK and a plan p_2 in Prometheus and the Prometheus p_2 performs the action a_1 . For instance, Withdraw Approved plan in Prometheus has an overlaps traceability relation with WithdrawApproved plan in JACK and Withdraw Approved plan in Prometheus performs Dispense Cash action (see Figure G.18). Therefore, there is a creates traceability relation between WithdrawApproved plan in JACK and Dispense Cash action in Prometheus.

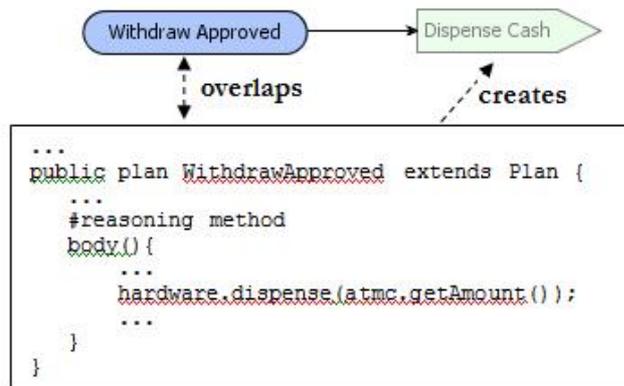


Figure G.18 JACK Plan vs Prometheus Action creates traceability relation

- JACK Plan vs Prometheus Message (*sends*) – a plan p_1 in JACK has a *sends* traceability relation with a message m_1 in Prometheus when there is an *overlaps* traceability relation between the plan p_1 in JACK and a plan p_2 in Prometheus and the Prometheus plan p_2 *sends* the message m_1 . For instance, Withdraw Cash plan in Prometheus has an overlaps traceability relation with WithdrawCash plan in JACK and Withdraw Cash plan in Prometheus sends Withdraw Request message (see Figure G.19). Therefore, there is a sends traceability relation between WithdrawCash plan in JACK and Withdraw Request message in Prometheus.

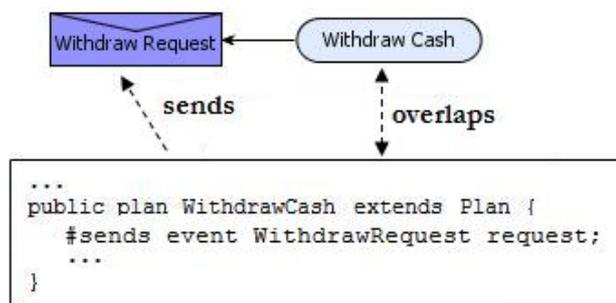


Figure G.19 JACK Plan vs Prometheus Message sends traceability relation

- JACK Plan vs Prometheus Message (*receives*) – a plan p_1 in JACK has *sends* traceability relation with a message m_1 in Prometheus when there is an *overlaps* traceability relation between the plan p_1 in JACK and a plan p_2 in Prometheus and the Prometheus plan p_2 *receives* the message m_1 . For instance, Withdraw Cash plan in Prometheus has an overlaps traceability relation with WithdrawCash plan in JACK and Withdraw Cash plan in Prometheus receives Withdraw message (see Figure G.20). Therefore, there is a receives traceability relation between WithdrawCash plan in JACK and Withdraw message in Prometheus.

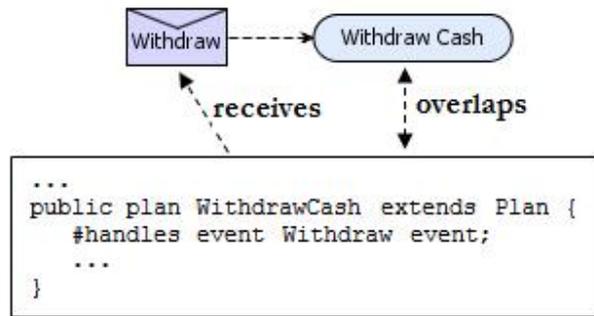


Figure G.20 JACK Plan vs Prometheus Message receives traceability relation

- JACK Plan vs Prometheus Data (*uses*) - a plan p_1 in JACK has *uses* traceability relation with a data d_1 in Prometheus when there is an *overlaps* traceability relation between the plan p_1 in JACK and a plan p_2 in Prometheus and the Prometheus plan p_2 *uses* the data d_1 . For instance, *Execute Advanced Search* plan in Prometheus has an *overlaps* traceability relation with *ExecuteAdvancedSearch* plan in JACK and *Execute Advanced Search* plan in Prometheus reads or modifies *BooksDB* data in Prometheus. Therefore, there is an *uses* traceability relation between *ExecuteAdvancedSearch* plan in JACK and *BooksDB* data in Prometheus. For instance, *Process Withdraw* plan in Prometheus and *ProcessWithdraw* plan in JACK has an *overlaps* traceability relation and *Process Withdraw* plan in Prometheus reads *accounts* data (see Figure G.21). Therefore, there is an *uses* traceability relation between *ProcessWithdraw* plan in JACK and *accounts* data in Prometheus.

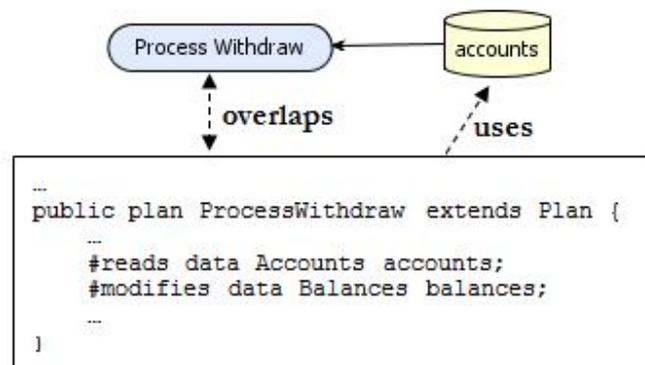


Figure G.21 JACK Plan vs Prometheus Data uses traceability relation

- JACK Plan vs Prometheus Data (*creates*) - a plan p_1 in JACK has *creates* traceability relation with a data d_1 in Prometheus when there is an *overlaps* traceability relation

between the plan p_1 in JACK and a plan p_2 in Prometheus and the Prometheus plan p_2 creates the data d_1 . For instance, Process Withdraw plan in Prometheus has an overlaps traceability relation with ProcessWithdraw plan in JACK and ProcessWithdraw plan writes on balances data (see Figure G.22). Therefore, there is a creates traceability relation between ProcessWithdraw plan in JACK and balances data in Prometheus.

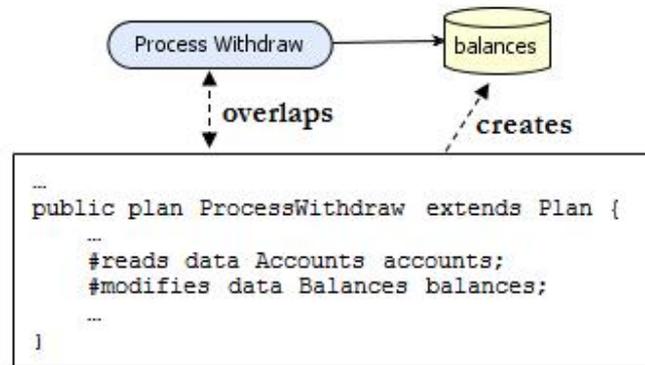


Figure G.22 JACK Plan vs Prometheus Data creates traceability relation

- JACK Belief vs Prometheus Role (*creates*) – a role r_1 in Prometheus has *creates* traceability relation with a beliefSet b_1 in Prometheus when there is an *overlaps* traceability relation between the beliefSet b_1 in Prometheus and a data d_2 in JACK and role r_1 writes on the beliefSet b_1 . For instance, *Customer Relation Management* role in Prometheus writes on *CustomerDB* data and *CustomerDB* data in Prometheus has an overlaps traceability relation with *CustomerDB* beliefSet in JACK (see Figure G.23). Therefore, there is a *creates* traceability relation between *Customer Relation Management* role in Prometheus and *CustomerDB* beliefSet in JACK.

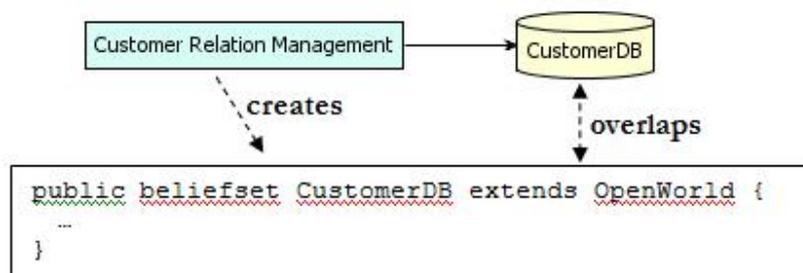


Figure G.23 JACK Belief vs Prometheus Role creates relation

- JACK Belief vs Prometheus Role (*uses*) – a role r_1 in Prometheus has *uses* traceability relation with a beliefSet b_1 in Prometheus when there is an *overlaps* traceability relation between the beliefSet b_1 in JACK and a data d_2 in Prometheus and the role r_1 reads the data d_1 . For instance, Purchasing role in Prometheus reads CustomerDB data and CustomerDB data has an overlaps traceability relation with CustomerDB beliefSet in JACK (see Figure G.24). Therefore, there is an uses traceability relation between Purchasing role and CustomerDB beliefSet in JACK.

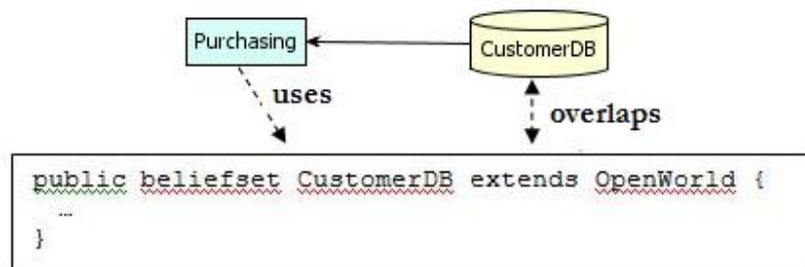


Figure G.24 JACK Belief vs Prometheus Role uses traceability relation

- JACK Belief vs Prometheus Agent (*creates*) – an agent a_1 in Prometheus has a *creates* traceability relation with a beliefSet b_1 in JACK when there is an *overlaps* traceability relation between the beliefSet b_1 in JACK and a data d_1 in Prometheus and the agent a_1 writes on the data d_1 . For instance, Atm agent in Prometheus writes on balances data and balances data in Prometheus has overlaps traceability with Balances beliefSet in JACK (see Figure G.25). Therefore, there is a creates traceability relation between Atm agent and Balances beliefSet.

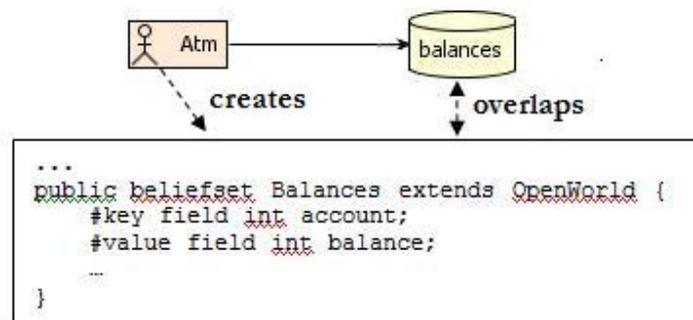


Figure G.25 JACK BeliefSet vs Prometheus Agent creates traceability relation

- JACK Belief vs Prometheus Agent (*uses*) – an agent a_1 in Prometheus has *uses* traceability relation with a beliefSet b_1 in Prometheus when there is an *overlaps* traceability relation between the data d_1 in Prometheus and a beliefSet b_2 in JACK and the agent a_1 in Prometheus *writes* on the data d_1 . For instance, Customer Relations agent in Prometheus *writes* on the CustomerDB data and there is an *overlaps* traceability relation between CustomerDB in Prometheus and CustomerDB in JACK. (see Figure G.26). Therefore, there is an *uses* traceability relation between Atm agent in Prometheus and Accounts beliefSet in JACK.

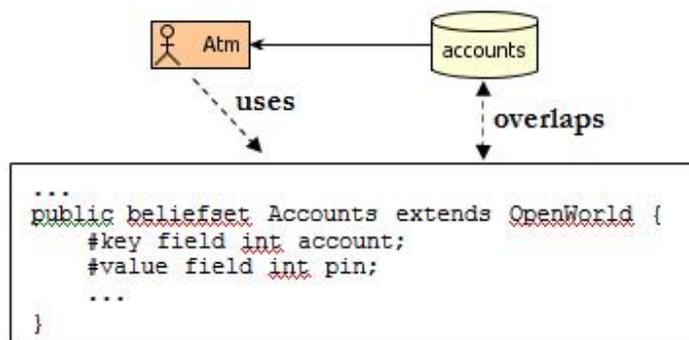


Figure G.26 JACK BeliefSet vs Prometheus Agent uses traceability relation

- JACK Belief vs Prometheus Capability (*creates*) – a capability c_1 in Prometheus has *creates* traceability relation with a beliefSet b_1 in Prometheus when the capability c_1 writes on a data d_1 and there is an *overlaps* traceability relation between the data d_1 in Prometheus and the beliefSet b_1 in JACK. For instance, ArrivalSequencing capability in Prometheus writes on runway_info data and runway_info data in Prometheus has an *overlaps* traceability relation with RunwayInfo beliefSet in JACK (see Figure G.27). Therefore, there is a *creates* traceability relation between ArrivalSequencing capability in Prometheus and RunwayInfo beliefSet in JACK.

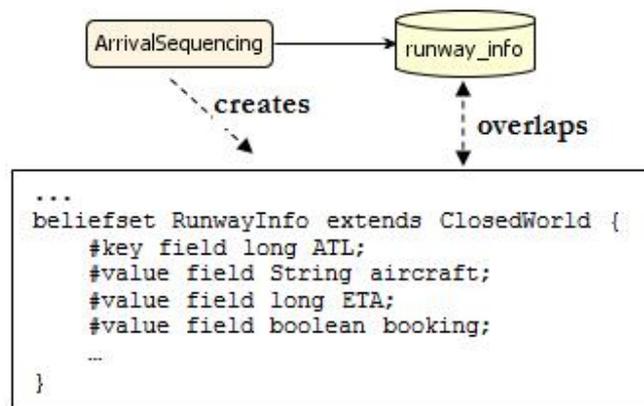


Figure G.27 JACK BeliefSet vs Prometheus Capability creates traceability relation

- JACK Belief vs Prometheus Capability (*uses*) – a capability c_1 in Prometheus has an *uses* traceability relation with a beliefSet b_1 in Prometheus when the capability c_1 reads a data d_1 and there is an *overlaps* traceability relation between the data d_1 in Prometheus and the beliefSet b_1 in JACK. For instance, ArrivalSequencing capability in Prometheus reads runway_info data and runway_info data in Prometheus has an overlaps traceability relation with RunwayInfo beliefSet in JACK (see Figure G.28). Therefore, there is a creates traceability relation between ArrivalSequencing capability in Prometheus and RunwayInfo beliefSet in JACK.

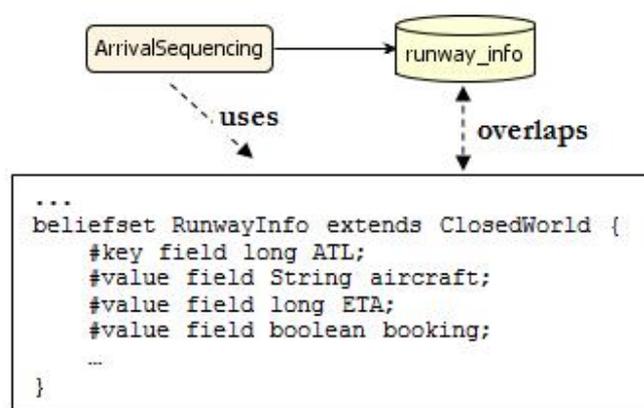


Figure G.28 JACK BeliefSet vs Prometheus Capability uses traceability relation

- JACK Belief vs Prometheus Plan (*creates*) – a plan p_1 in Prometheus has *creates* traceability relation with a beliefSet b_1 in JACK when there is an *overlaps* traceability relation between the beliefSet b_1 in JACK and a data d_1 in Prometheus and the plan p_1 in Prometheus writes on the data d_1 . For instance, Process Withdraw plan in Prometheus writes on balances data and balances data in Prometheus has an overlaps traceability relation with Balances beliefSet in JACK (see Figure G.29). Therefore, there is a creates traceability relation between Process Withdraw plan in Prometheus and Balances beliefSet in JACK.

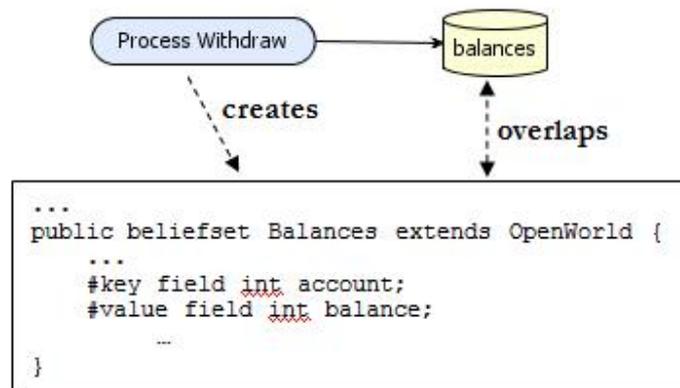


Figure G.29 JACK BeliefSet vs Prometheus Plan creates traceability relation

- JACK Belief vs Prometheus Plan (*uses*) – a plan p_1 in Prometheus has *uses* traceability relation with a beliefSet b_1 in Prometheus when there is an *overlaps* traceability relation between the beliefSet b_1 in JACK and a data d_1 in Prometheus and the plan p_1 in Prometheus reads the data d_1 . For instance, Process Withdraw plan in Prometheus reads balances data and balance data in Prometheus has an overlaps traceability relation with Balances beliefSet in JACK (see Figure G.30). Therefore, there is an uses traceability relation between Process Withdraw plan in Prometheus and Balances beliefSet in JACK.

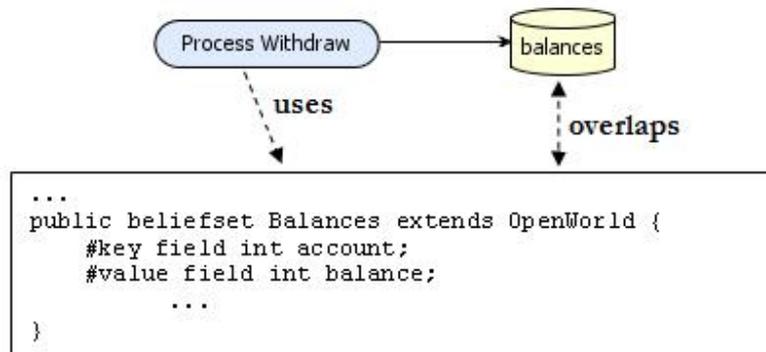


Figure G.30 JACK BeliefSet vs Prometheus uses traceability relation

- JACK BeliefSet vs Prometheus Data - A data in Prometheus and a beliefSet in JACK has an *overlaps* traceability relation when the name of the data is synonyms to the name of the beliefSet and if the name of the the fields of the data and the beliefset are similar (see Figure G.31). For instance, the name of accounts data in Prometheus is synonyms to the name of Accounts beliefSet in JACK and the fieds of the Accounts beliefSet (account and pin) are similar to the fields of the accounts data in Prometheus.

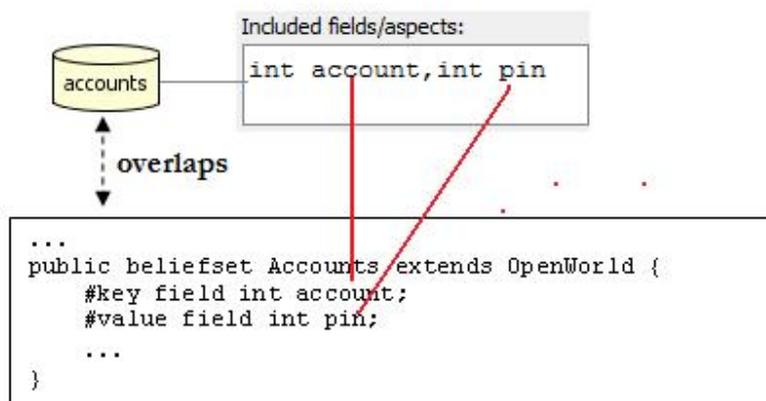


Figure G.31 JACK BeliefSet vs Prometheus Data overlaps traceability relation

- JACK Event vs Prometheus Agent (*receives*) – an agent a_1 in Prometheus has a *receives* traceability relation with an event e_1 in JACK when there is a message m_1 in Prometheus that has an *overlaps* traceability relation with an event e_1 and the Prometheus agent a_1 receives the message m_1 . For instance, Bank agent in Prometheus receives Withdraw Request message and Withdraw Request message in Prometheus

has an overlaps traceability relation with WithdrawRequest event in JACK (see Figure G.32). Therefore, there is a receives traceability relation between Bank agent in Prometheus and WithdrawRequest event in JACK.

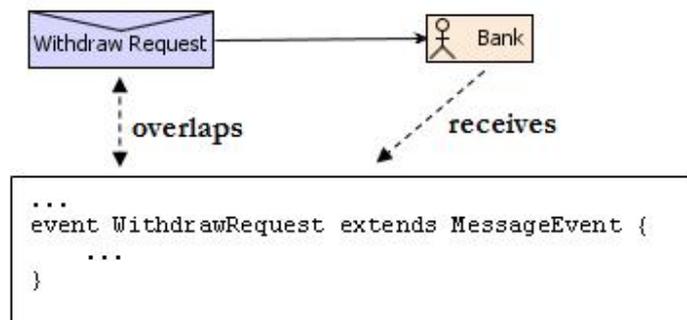


Figure G.32 JACK Event vs Prometheus Agent receives traceability relation

- JACK Event vs Prometheus Agent (*sends*) – an agent a_1 in Prometheus has a *sends* traceability relation with an event e_1 in JACK when there is a message m_1 in Prometheus that has an *overlaps* traceability relation with an event e_1 and the Prometheus agent a_1 sends the message m_1 . For instance, Bank agent in Prometheus sends Withdraw Response message and Withdraw Response message in Prometheus has an overlaps traceability relation with WithdrawResponse event in JACK (see Figure G.33). Therefore, there is a sends traceability relation between Bank agent in Prometheus and WithdrawRequest event in JACK

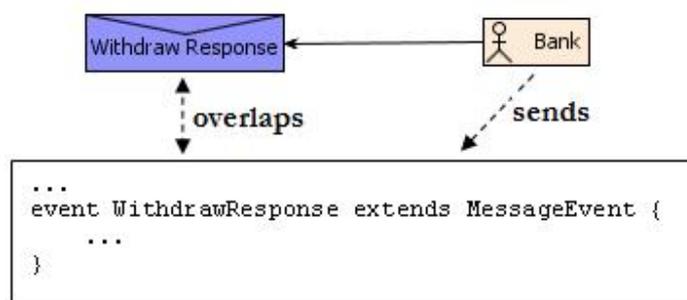


Figure G.33 JACK Event vs Prometheus Agent sends traceability relation

- JACK Event vs Prometheus Capability (*sends*) – a capability c_1 in Prometheus has a *sends* traceability relation with an event e_1 in JACK when there is a message m_1 that has an *overlaps* traceability relation with an event e_1 and the capability c_1 includes the

message m_1 . For instance, Traffic Feeding capability in Prometheus sends AircraftEvent message and AircraftEvent message in Prometheus has an overlaps traceability relation with AircraftEvent in JACK (see Figure G.34). Therefore, there is a sends traceability relation between Traffic Feeding capability in Prometheus and AircraftEvent in JACK.

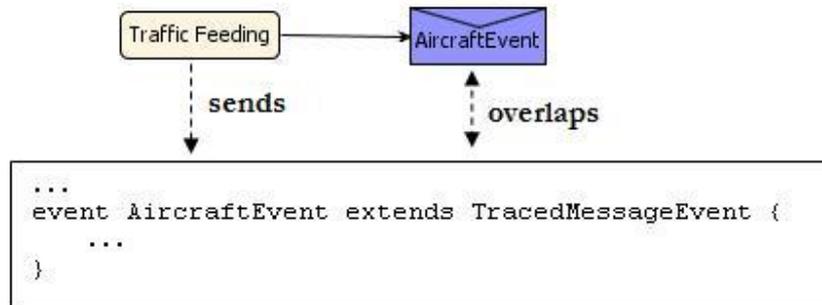


Figure G.34 JACK Event vs Prometheus Capability

- JACK Event vs Prometheus Capability (receives) – a capability c_1 in Prometheus has a *receives* traceability relation with an event e_1 in JACK when there is a message m_1 that has an *overlaps* traceability relation with an event e_1 and the capability c_1 receives the message m_1 . For instance, Flying capability in Prometheus receives Enter Control Area message and Enter Control Area message in Prometheus has an overlaps traceability relation with EnterControlArea event in JACK (see Figure G.35). Therefore, there is a receives traceability relation between Flying capability in Prometheus and EnterControlArea event in JACK.

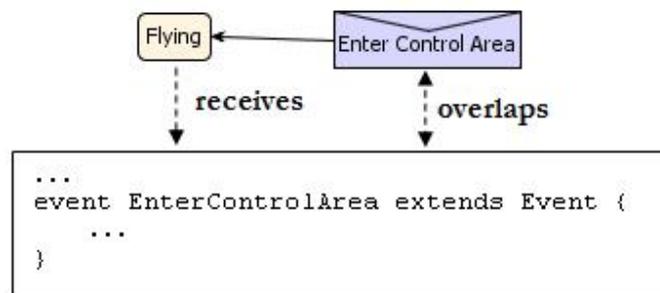


Figure G.35 JACK Event vs Prometheus Capability receives relation

- JACK Event vs Prometheus Plan (sends) – a plan p_1 in Prometheus has a *sends* traceability relation with an event e_1 in JACK when there is a message m_1 that has an *overlaps* traceability relation with an event e_1 and the plan p_1 sends the message m_1 . For instance, Process Withdraw plan in Prometheus sends Withdraw Response message and Withdraw Response message in Prometheus has an overlaps traceability relation with WithdrawResponse event in JACK (see Figure G.36). Therefore, there is a sends traceability relation between Process Withdraw plan in Prometheus and WithdrawResponse event in JACK.

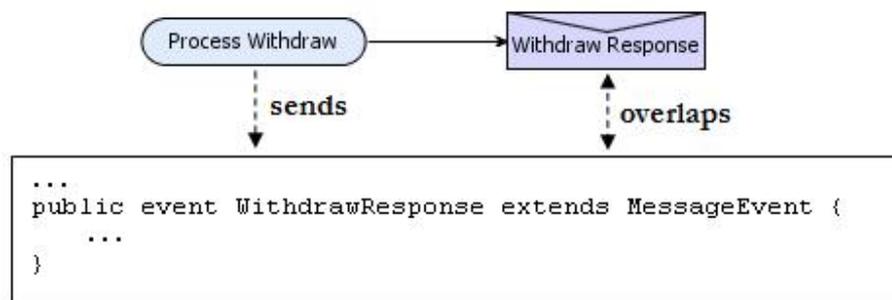


Figure G.36 JACK Event vs Prometheus Plan sends traceability relation

- JACK Event vs Prometheus Plan (receives) – a plan p_1 in Prometheus has a *receives* traceability relation with an event e_1 in JACK when there is a message m_1 that has an *overlaps* traceability relation with an event e_1 and the plan p_1 includes the message m_1 . For instance, Process Withdraw plan in Prometheus receives Withdraw Request message and Withdraw Request message in Prometheus has an overlaps traceability relation with WithdrawRequest event in JACK (see Figure G.37). Therefore, there is an receives traceability relation between Process Withdraw plan in Prometheus and WithdrawRequest event in JACK.

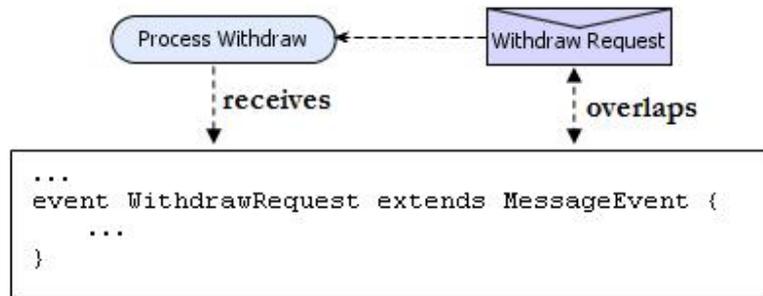


Figure G.37 JACK Event vs Prometheus Plan receives traceability relation

- JACK Event vs Prometheus Message (*overlaps*) – an event e_1 in JACK have an *overlaps* traceability relation with a message m_1 if the name of type of the event is *synonyms* to the name of the message in Prometheus. For instance, Withdraw Request message in Prometheus has a synonyms name to WithdrawRequest event in JACK (see Figure G.38). Therefore, there is an overlaps traceability relation between Withdraw Request message and WithdrawRequest event in JACK.

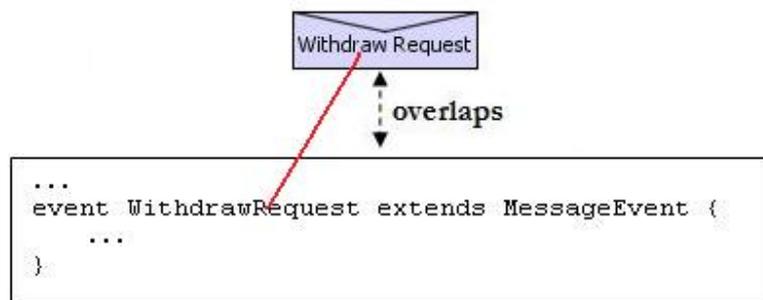


Figure G.38 JACK Event vs Prometheus Message overlaps traceability relation