**SQWRL Expressions, descriptions, and results**

|  |  |  |
| --- | --- | --- |
| **SQWRL Expression** | **Description** | **Result** |
| State(?s) ^ isActive(?s, true) -> sqwrl:select(?s) | Verifies which states are active dynamically. | (Correct)  Active State: :State-6  Active State: :State-7 |
| CurrentState(?cs) -> sqwrl:select(?cs) | Verifies which states are currently active dynamically. | (Correct)  Current State: :State-6  Current State: :State-7 |
| NextState(?ns) -> sqwrl:select(?ns) | Verifies inference which states are next states of the current state(s) dynamically. | (Correct)  Next State: :State-0 |
| PreviousState(?ps) -> sqwrl:select(?ps) | Verifies which states are previous states of the current state(s) dynamically. | (Correct)  Previous State: :State-5 |
| Thing(?x) -> sqwrl:selectDistinct(?x) | Verifies which agents are Things in the system. | (Correct)  I am a Thing: :Device-I |
| Agent(?a) ^ Service(?s) ^ offers(?a, ?s) -> sqwrl:select(?a, ?s) | Verifies which services offer each agent in the system. | (Correct)  I am: :Server-I and I offer: :Save-Data-Service  I am: :Device-I and I offer: :Send-Piece-Data-Service  I am: :Device-II and I offer: :Perform-Separation-Service |
| Standard(?s) ^ hasEmbeddingCapability(?s, ?c) -> sqwrl:select(?s, ?c) | Verifies which standards are tagged with None, Low, Medium, or High embedding capability in the system. | (Correct)  Standard: :Standard-IEC-61499 | Embedding capability: "High"^^xsd:string  Standard: :Standard-OPC-UA | Embedding capability: "High"^^xsd:string  Standard: :Standard-IEC-62890 | Embedding capability: "Low"^^xsd:string  Standard: :Standard-IEC-62264-ISA-95 | Embedding capability: "High"^^xsd:string  Standard: :Standard-IEC-61512-ISA-88 | Embedding capability: "Medium"^^xsd:string  Standard: :Standard-IEC-61131 | Embedding capability: "High"^^xsd:string  Standard: :Standard-W3C | Embedding capability: "High"^^xsd:string |
| Sequence(?s) ^ isParentSetElement(?s, true) -> sqwrl:select(?s) | Verifies which sequences are parent (main) sequences in the system. | (Correct)  Parent Sequence: :Metal-Separation-Sequence |
| Agent(?a) ^ belongsToArchitectureLayer(?a, ?lvl) -> sqwrl:selectDistinct(?a, ?lvl) | Verifies the level of the ISA 95 architecture that each agent in the system belongs to. | (Correct)  Actor :Device-I located in : "1"^^xsd:integer architecture level  Actor :Device-I located in : "3"^^xsd:integer architecture level  Actor :Device-II located in : "2"^^xsd:integer architecture level  Actor :Server-I located in : "4"^^xsd:integer architecture level |
| Agent(?a) ^ hasDescription(?a, ?d) ^ swrlb:contains(?d, "controller") -> sqwrl:select(?a) | Consists of verifying by means of inference which agents contains the word ‘controller’ in the description. | (Correct)  Actor :Device-II contains 'controller' in description. |
| ConcurrentState(?cs) -> sqwrl:select(?cs) | Consists of verifying by means of inference which states run concurrently or in parallel execution dynamically. | (Incorrect)  No result. |
| Device(?x) -> sqwrl:selectDistinct(?x) | Consists of verifying by means of inference which agents are Devices in the system. | (Correct)  I am a Device : :Device-II |
| FinalState(?fs) -> sqwrl:select(?fs) | Consists of verifying by means of inference which states are final states in the system. | (Correct)  Final State : :State-7  Final State : :State-6 |
| hasInteroperabilityDegree(?x, "High") ^ Agent(?x) -> sqwrl:selectDistinct(?x) | Consists of verifying by means of inference which agents are tagged with a High interoperability degree in the system. | (Correct)  Actor with high interoperability degree : :Device-I  Actor with high interoperability degree : :Device-II  Actor with high interoperability degree : :Server-I |
| InitialState(?is) -> sqwrl:select(?is) | Consists of verifying by means of inference which states are initial states in the system. | (Correct)  Initial state : :State-0 |
| Service(?s) ^ hasDescription(?s, ?d) ^ swrlb:contains(?d, "database") -> sqwrl:select(?s) | Consists of verifying by means of inference which services contains the word ‘database’ in the description. | (Correct)  Service : :Save-Data-Service contains 'database' in service description. |
| SoftwareResource(?x) -> sqwrl:selectDistinct(?x) | Consists of verifying by means of inference which actors are Software Resources in the system. | (Correct)  I am a Software Resource : :Device-I  I am a Software Resource : :Server-I |
| SynchronousState(?ss) -> sqwrl:select(?ss) | Consists of verifying by means of inference which states are synchronous states in the system dynamically. | (Incorrect)  No result. |
| Standard(?s) ^ Device(?d) ^ standardizes(?s, ?d) -> sqwrl:selectDistinct(?s) | Consists of verifying by means of inference which standards concern to devices in the system. | (Correct)  Standard : :Standard-IEC-61131 standardizes devices  Standard : :Standard-IEC-61499 standardizes devices  Standard : :Standard-IEC-62264-ISA-95 standardizes devices  Standard : :Standard-OPC-UA standardizes devices |
| interacts(?x, ?y) ^ Actor(?y) ^ Actor(?x) -> sqwrl:selectDistinct(?x, ?y) | Consists of verifying by means of inference which actors communicate or interact one another. | (Correct)  Actor :Device-I interacts with Actor :Device-II  Actor :Device-I interacts with Actor :Server-I  Actor :Server-I interacts with Actor :Device-I |
| Agent(?a) ^ hasFeature(?a, ?f) ^ swrlb:equal(?f, "Proactive") -> sqwrl:selectDistinct(?a) | Queries distinct agents which are classified as ‘proactive’ in the system. |  |
| Agent(?a) ^ hasFeature(?a, ?f) ^ swrlb:equal(?f, "Reactive") -> sqwrl:selectDistinct(?a) | Queries distinct agents which are classified as ‘reactive’ in the system. |  |

**Associated code:**

|  |  |
| --- | --- |
| 1 | *SQWRLResult* q0Result = queryEngine.runSQWRLQuery("QueryActiveStates"); |
| 2 |  |
| 3 | *String* messageResult = "Active State Query Inference Result:\n"; |
| 4 | while (q0Result.next()) { |
| 5 | messageResult += "Active State: " + q0Result.getValue("s") + "\n"; |
| 6 | } |
| 7 |  |
| 8 | *SQWRLResult* q1Result = queryEngine.runSQWRLQuery("QueryCurrentStates"); |
| 9 |  |
| 10 | messageResult = "Current State Query Inference Result:\n"; |
| 11 |  |
| 12 | while (q1Result.next()) { |
| 13 | messageResult += "Current State: " + q1Result.getValue("cs") + "\n"; |
| 14 | } |
| 15 |  |
| 16 | messageResult = "Next State Query Inference Result:\n"; |
| 17 |  |
| 18 | *SQWRLResult* q2Result = queryEngine.runSQWRLQuery("QueryNextState"); |
| 19 | while (q2Result.next()) { |
| 20 | messageResult += "Next State: " + q2Result.getValue("ns") + "\n"; |
| 21 | } |
| 22 |  |
| 23 | messageResult = "Previous State Query Inference Result:\n"; |
| 24 |  |
| 25 | *SQWRLResult* q3Result = queryEngine.runSQWRLQuery("QueryPreviousState"); |
| 26 | while (q3Result.next()) { |
| 27 | messageResult += "Previous State: " + q3Result.getValue("ps") + "\n"; |
| 28 | } |
| 29 |  |
| 30 | messageResult = "Things Query Inference Result:\n"; |
| 31 |  |
| 32 | *SQWRLResult* q5Result = queryEngine.runSQWRLQuery("QueryThings"); |
| 33 | while (q5Result.next()) { |
| 34 | messageResult += "I am a Thing: " + q5Result.getValue("x") + "\n"; |
| 35 | } |
| 36 |  |
| 37 | messageResult = "Services by Agent Query Inference Result:\n"; |
| 38 |  |
| 39 | *SQWRLResult* q6Result = queryEngine.runSQWRLQuery("QueryServicesByAgent"); |
| 40 | while (q6Result.next()) { |
| 41 | messageResult += "I am: " + q6Result.getValue("a") + " and I offer: " + q6Result.getValue("s") + "\n"; |
| 42 | } |
| 43 |  |
| 44 | messageResult = "Standards and Embedding Capability Query Inference Result:\n"; |
| 45 |  |
| 46 | *SQWRLResult* q7Result = queryEngine.runSQWRLQuery("QueryStandardAndEmbeddingCapability"); |
| 47 | while (q7Result.next()) { |
| 48 | messageResult += "Standard: " + q7Result.getValue("s") + " | Embedding capability: " + q7Result.getValue("c") + "\n"; |
| 49 | } |
| 50 |  |
| 51 | messageResult = "Parent Sequences Query Inference Result:\n"; |
| 52 |  |
| 53 | *SQWRLResult* q8Result = queryEngine.runSQWRLQuery("QueryParentSequences"); |
| 54 | while (q8Result.next()) { |
| 55 | messageResult += "Parent Sequence: " + q8Result.getValue("s") + "\n"; |
| 56 | } |
| 57 |  |
| 58 | messageResult = "Actor Architecture Level Query Inference Result:\n"; |
| 59 |  |
| 60 | *SQWRLResult* q9Result = queryEngine.runSQWRLQuery("QueryAgentArchitectureLevel"); |
| 61 | while (q9Result.next()) { |
| 62 | messageResult += "Actor " + q9Result.getValue("a") + " located in : " + q9Result.getValue("lvl") + " architecture level\n"; |
| 63 | } |
| 64 |  |
| 65 | messageResult = "Agent with 'controller' Description Query Inference Result:\n"; |
| 66 |  |
| 67 | *SQWRLResult* q10Result = queryEngine.runSQWRLQuery("QueryAgentWithControllerDesc"); |
| 68 | while (q10Result.next()) { |
| 69 | messageResult += "Agent " + q10Result.getValue("a").toString() + " contains 'controller' in description\n"; |
| 70 | } |
| 71 |  |
| 72 | messageResult = "Concurrent State Query Inference Result:\n"; |
| 73 |  |
| 74 | *SQWRLResult* q11Result = queryEngine.runSQWRLQuery("QueryConcurrentStates"); |
| 75 | while (q11Result.next()) { |
| 76 | messageResult += "Concurrent State : " + q11Result.getValue("cs").toString() + "\n"; |
| 77 | } |
| 78 |  |
| 79 | messageResult = "Devices Query Inference Result:\n"; |
| 80 |  |
| 81 | *SQWRLResult* q12Result = queryEngine.runSQWRLQuery("QueryDevices"); |
| 82 | while (q12Result.next()) { |
| 83 | messageResult += "I am a Device : " + q12Result.getValue("x").toString() + "\n"; |
| 84 | } |
| 85 |  |
| 86 | messageResult = "Final State Query Inference Result:\n"; |
| 87 |  |
| 88 | *SQWRLResult* q13Result = queryEngine.runSQWRLQuery("QueryFinalStates"); |
| 89 | while (q13Result.next()) { |
| 90 | messageResult += "Final State : " + q13Result.getValue("fs").toString() + "\n"; |
| 91 | } |
| 92 |  |
| 93 | messageResult = "Agents with High Interoperability Degree Query Inference Result:\n"; |
| 94 |  |
| 95 | *SQWRLResult* q14Result = queryEngine.runSQWRLQuery("QueryForAgentsWithHighInteroperabilityDegree"); |
| 96 | while (q14Result.next()) { |
| 97 | messageResult += "Agent with high interoperability degree : " + q14Result.getValue("x").toString() + "\n"; |
| 98 | } |
| 99 |  |
| 100 | messageResult = "Initial State Query Inference Result:\n"; |
| 101 |  |
| 102 | *SQWRLResult* q15Result = queryEngine.runSQWRLQuery("QueryInitialStates"); |
| 103 | while (q15Result.next()) { |
| 104 | messageResult += "Initial state : " + q15Result.getValue("is").toString() + "\n"; |
| 105 | } |
| 106 |  |
| 107 | messageResult = "Services with 'database' Description Query Inference Result:\n"; |
| 108 |  |
| 109 | *SQWRLResult* q16Result = queryEngine.runSQWRLQuery("QueryServicesWithDatabaseContent"); |
| 110 | while (q16Result.next()) { |
| 111 | messageResult += "Service : " + q16Result.getValue("s").toString() + " contains 'database' in service description\n"; |
| 112 | } |
| 113 |  |
| 114 | messageResult = "Software Resources Query Inference Result:\n"; |
| 115 |  |
| 116 | *SQWRLResult* q17Result = queryEngine.runSQWRLQuery("QuerySoftwareResource"); |
| 117 | while (q17Result.next()) { |
| 118 | messageResult += "I am a Software Resource : " + q17Result.getValue("x").toString() + "\n"; |
| 119 | } |
| 120 |  |
| 121 | messageResult = "Synchronous State Query Inference Result:\n"; |
| 122 |  |
| 123 | *SQWRLResult* q19Result = queryEngine.runSQWRLQuery("QuerySynchronousStates"); |
| 124 | while (q19Result.next()) { |
| 125 | messageResult += "Synchronous State : " + q19Result.getValue("ss").toString() + "\n"; |
| 126 | } |
| 127 |  |
| 128 | messageResult = "Standards Which Concern to Devices Query Inference Result:\n"; |
| 129 |  |
| 130 | *SQWRLResult* q20Result = queryEngine.runSQWRLQuery("QueryWhatStandardsStandardizeDevices"); |
| 131 | while (q20Result.next()) { |
| 132 | messageResult += "Standard : " + q20Result.getValue("s").toString() + " standardizes devices\n"; |
| 133 | } |
| 134 |  |
| 135 | messageResult = "Interaction Pairs (Actors) Query Inference Result:\n"; |
| 136 |  |
| 137 | *SQWRLResult* q21Result = queryEngine.runSQWRLQuery("QueryForInteractionPairs"); |
| 138 | while (q21Result.next()) { |
| 139 | messageResult += "Actor " + q21Result.getValue("x").toString() + " interacts with Actor " + q21Result.getValue("y").toString() + "\n"; |
| 140 | } |