ABRTTDMS MSDs-generated pi-calculus code

Aissam Belghiat

January 22, 2026

1 Pi-Calculus Formalization the system of MIC State Machine

1.1 System Overview

The Mobile Information Collector (MIC) system is a mobile agent-based software system consisting of one mobile agent MIC that interacts with a stationary agent TMCA (Traffic Management Coordinating Authority). The agent operates across five places: P1 (TMCA base), P2 (Police Station), P3 (Emergency Services), P4 (Maintenance Depot), and P1 again for return.

Let $MASS_{MIC} = (MIC\text{-}System, P, MSD_{MIC}, MSD_{TMCA})$ where:

- $P = \{P1, P2, P3, P4\}$: the set of places
- MSD_{MIC}: the mobile state-transition diagram for the MIC agent
- MSD_{TMCA}: the stationary agent at place P1

1.2 Channel Definitions

We define the following shortcuts for simplification:

$$\vec{l} \stackrel{def}{=} l1, l2, l3, l4$$
$$\vec{p} \stackrel{def}{=} p1, p2, p3, p4$$
$$\vec{evext} \stackrel{def}{=} evext1, evext2, evext3, evext4, evext5$$
$$\vec{remote} \stackrel{def}{=} remote1, remote2, remote3, remote4, remote5$$

Expected events for MIC agent:

$$\vec{e}_{MIC} \stackrel{def}{=} \text{createAgent, dispatchCommand, networkFailure, retryTimerExpired}$$

Expected conditions for MIC agent:

$$\begin{aligned}\vec{c}_{MIC} \stackrel{def}{=}\ & \text{authenticationSuccess, authenticationFailure, networkRestored, atMigrationStep1,}\\ & \text{atMigrationStep2, atMigrationStep3, atMigrationStepReturn,}\\ & \text{timeoutExceeded, maxRetriesExceeded, atLocationPolice,}\\ & \text{atLocationEmergency, atLocationMaintenance, completenessOK, consistencyOK,}\\ & \text{maxAuthRetriesExceeded, allValidationsComplete}\end{aligned}$$

Expected actions for MIC agent:

$$\vec{act}_{MIC} \stackrel{def}{=} \text{arrivedAtPolice, arrivedAtEmergency, arrivedAtMaintenance, arrivedAtTMCA,}$$
$$\text{missionStep1Complete, missionStep2Complete, missionStep3Complete, prepareFailureReport,}$$
$$\text{queryExecuted, dataRetrieved, packagingComplete, progressUpdate,}$$
$$\text{results, mergeComplete, releaseResources}$$

1.3 Place Processes

1.4 Place Processes

Following Rule 3, the behavior of place $P1$ is specified by:

$$\begin{aligned}
P1_0(\vec{l},\vec{p}) &\stackrel{def}{=} p1(a1,m1).P1_1(\vec{l},\vec{p},a1,m1)\\
P1_1(\vec{l},\vec{p},a1,m1) &\stackrel{def}{=} \overline{a1}.P1_1(\vec{l},\vec{p},a1,m1) + p1(a2,m2).P1_2(\vec{l},\vec{p},a1,m1,a2,m2)+\\
&\quad m1(d).([d=l2]\overline{p2}\langle a1,m1\rangle.P1_0(\vec{l},\vec{p}) + [d=l3]\overline{p3}\langle a1,m1\rangle.P1_0(\vec{l},\vec{p})+\\
&\quad [d=l4]\overline{p4}\langle a1,m1\rangle.P1_0(\vec{l},\vec{p}))\\
P1_2(\vec{l},\vec{p},a1,m1,a2,m2) &\stackrel{def}{=} \overline{a1}.P1_2(\vec{l},\vec{p},a1,m1,a2,m2) + \overline{a2}.P1_2(\vec{l},\vec{p},a1,m1,a2,m2)+\\
&\quad m1(d).([d=l2]\overline{p2}\langle a1,m1\rangle.P1_1(\vec{l},\vec{p},a2,m2)+\\
&\quad [d=l3]\overline{p3}\langle a1,m1\rangle.P1_1(\vec{l},\vec{p},a2,m2)+\\
&\quad [d=l4]\overline{p4}\langle a1,m1\rangle.P1_1(\vec{l},\vec{p},a2,m2))+\\
&\quad m2(d).([d=l2]\overline{p2}\langle a2,m2\rangle.P1_1(\vec{l},\vec{p},a1,m1)+\\
&\quad [d=l3]\overline{p3}\langle a2,m2\rangle.P1_1(\vec{l},\vec{p},a1,m1)+\\
&\quad [d=l4]\overline{p4}\langle a2,m2\rangle.P1_1(\vec{l},\vec{p},a1,m1))
\end{aligned}$$

Similarly for places $P2$, $P3$, and $P4$:

$$\begin{aligned}
P2_0(\vec{l},\vec{p}) &\stackrel{def}{=} p2(a1,m1).P2_1(\vec{l},\vec{p},a1,m1)\\
P2_1(\vec{l},\vec{p},a1,m1) &\stackrel{def}{=} \overline{a2}.P2_1(\vec{l},\vec{p},a1,m1)+\\
&\quad m1(d).([d=l1]\overline{p1}\langle a1,m1\rangle.P2_0(\vec{l},\vec{p}) + [d=l3]\overline{p3}\langle a1,m1\rangle.P2_0(\vec{l},\vec{p})+\\
&\quad [d=l4]\overline{p4}\langle a1,m1\rangle.P2_0(\vec{l},\vec{p}))
\end{aligned}$$

$$\begin{aligned}
P3_0(\vec{l},\vec{p}) &\stackrel{def}{=} p3(a1,m1).P3_1(\vec{l},\vec{p},a1,m1)\\
P3_1(\vec{l},\vec{p},a1,m1) &\stackrel{def}{=} \overline{a3}.P3_1(\vec{l},\vec{p},a1,m1)+\\
&\quad m1(d).([d=l1]\overline{p1}\langle a1,m1\rangle.P3_0(\vec{l},\vec{p}) + [d=l2]\overline{p2}\langle a1,m1\rangle.P3_0(\vec{l},\vec{p})+\\
&\quad [d=l4]\overline{p4}\langle a1,m1\rangle.P3_0(\vec{l},\vec{p}))
\end{aligned}$$

$$P4_0(\vec{l},\vec{p}) \stackrel{def}{=} p4(a1,m1).P4_1(\vec{l},\vec{p},a1,m1)$$

$$\begin{aligned} P4_1(\vec{l},\vec{p},a1,m1) \stackrel{def}{=}\ & \overline{a4}.P4_1(\vec{l},\vec{p},a1,m1)+ \\ & m1(d).([d=l1]\overline{p1}\langle a1,m1\rangle.P4_0(\vec{l},\vec{p}) + [d=l2]\overline{p2}\langle a1,m1\rangle.P4_0(\vec{l},\vec{p})+ \\ & [d=l3]\overline{p3}\langle a1,m1\rangle.P4_0(\vec{l},\vec{p})) \end{aligned}$$

Following Rule 4, the initial place configuration is:

$$Pc(\vec{l},a1,m1) \stackrel{def}{=} (\nu\vec{p})(P1_2(\vec{l},\vec{p},a1,m1,a2,m2) \mid P2_0(\vec{l},\vec{p}) \mid P3_0(\vec{l},\vec{p}) \mid P4_0(\vec{l},\vec{p}))$$

1.5 Pseudo-State Processes

Following Rules 1 and 2:

$$I_{MIC}(createagent) \stackrel{def}{=} createagent.Created$$

$$F_{MIC}(end) \stackrel{def}{=} \overline{end}.F_{MIC}(end)$$

1.6 Simple State Processes

Following Rule 5, the simple states are defined as:

$$Created(a1) \stackrel{def}{=} a1.Created(a1)$$

$$MergingResults(a1) \stackrel{def}{=} a1.MergingResults(a1)$$

$$CreatingFinalReport(a1) \stackrel{def}{=} a1.CreatingFinalReport(a1)$$

$$ReportingToTMCA(a1) \stackrel{def}{=} a1.ReportingToTMCA(a1)$$

$$Terminated(a1) \stackrel{def}{=} a1.Terminated(a1)$$

$$WaitingForNetwork(a1) \stackrel{def}{=} a1.WaitingForNetwork(a1)$$

$$RetryingMigration(a1) \stackrel{def}{=} a1.RetryingMigration(a1)$$

$$AbortingMission(a1) \stackrel{def}{=} a1.AbortingMission(a1)$$

$$WaitingRetryAuth(a1) \stackrel{def}{=} a1.WaitingRetryAuth(a1)$$

1.7 Mobile State Processes

Following Rule 5, mobile states are defined as:

$$TravelingToPolice(a1) \stackrel{def}{=} a1.TravelingToPolice(a1)$$
$$TravelingToEmergency(a1) \stackrel{def}{=} a1.TravelingToEmergency(a1)$$
$$TravelingToMaintenance(a1) \stackrel{def}{=} a1.TravelingToMaintenance(a1)$$
$$ReturningToTMCA(a1) \stackrel{def}{=} a1.ReturningToTMCA(a1)$$

1.8 State Processes with Transitions

1.8.1 Created State with Event Transition

Following Rules 5 and 11, the Created state receives the DispatchCommand event and transitions to TravelingToPolice:

$$\begin{aligned} Created(a1, m1, \vec{l}, evint1, rtc1, \vec{e}_{MIC}) \stackrel{def}{=}\ & a1.Created(a1, m1, \vec{l}, evint1, rtc1, \vec{e}_{MIC}) + \\ & evint1(z).([z = \text{dispatchCommand}]\overline{rtc1}.TravelingToPolice(a1, m1, \vec{l}, evint1, rtc1, \vec{e}_{MIC}) + \\ & \sum_{i \neq \text{dispatchCommand}} [z = e_i]\overline{rtc1}.Created(a1, m1, \vec{l}, evint1, rtc1, \vec{e}_{MIC})) \end{aligned}$$

1.8.2 TravelingToPolice State with Action and Failure Transitions

Following Rules 7, 11, and 16:

$$\begin{aligned} TravelingToPolice(a1, m1, \vec{l}, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC}) \stackrel{def}{=}\ & \\ & a1.TravelingToPolice(a1, m1, \vec{l}, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC}) + \\ & evint1(z).([z = \text{networkFailure}]\overline{rtc1}.WaitingForNetwork(a1, m1, \vec{l}, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC}) + \\ & \sum_{i \neq \text{networkFailure}} [z = e_i]\overline{rtc1}.TravelingToPolice(a1, m1, \vec{l}, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC})) + \\ & \overline{m1}\langle l2\rangle.(\nu done1)(AH_{\text{arrivedAtPolice}}(\text{arrivedAtPolice}, done1) \mid \\ & \quad done1.PoliceStationOperations(a1, m1, \vec{l}, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC})) \end{aligned}$$

where the action handler is:

$$AH_{\text{arrivedAtPolice}}(\text{arrivedAtPolice}, done1) \stackrel{def}{=} \overline{arrivedAtPolice}.\overline{done1}.0$$

Similar formalization for TravelingToEmergency, TravelingToMaintenance, ReturningToTMCA:

$$\begin{aligned} TravelingToEmergency(a1, m1, \vec{l}, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC}) \stackrel{def}{=}\ & \\ & a1.TravelingToEmergency(a1, m1, \vec{l}, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC}) + \\ & evint1(z).([z = \text{networkFailure}]\overline{rtc1}.WaitingForNetwork(a1, m1, \vec{l}, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC}) + \\ & \sum_{i \neq \text{networkFailure}} [z = e_i]\overline{rtc1}.TravelingToEmergency(a1, m1, \vec{l}, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC})) + \\ & \overline{m1}\langle l3\rangle.(\nu done1)(AH_{\text{arrivedAtEmergency}}(\text{arrivedAtEmergency}, done1) \mid \\ & \quad done1.EmergencyServicesOperations(a1, m1, \vec{l}, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC})) \end{aligned}$$

$$AH_{\text{arrivedAtEmergency}}(\text{arrivedAtEmergency}, done1) \stackrel{def}{=} \overline{arrivedAtEmergency}.\overline{done1}.0$$

$$\begin{aligned}
TravelingToMaintenance(a1, m1, \vec{l}, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC}) \stackrel{def}{=} \\
& a1.TravelingToMaintenance(a1, m1, \vec{l}, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC}) + \\
& evint1(z).([z = \text{networkFailure}]\overline{rtc1}.WaitingForNetwork(a1, m1, \vec{l}, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC}) + \\
& \sum_{i \neq \text{networkFailure}} [z = e_i]\overline{rtc1}.TravelingToMaintenance(a1, m1, \vec{l}, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC})) + \\
& \overline{m1}\langle l4\rangle.(\nu done1)(AH_{\text{arrivedAtMaintenance}}(\text{arrivedAtMaintenance}, done1) \mid \\
& \quad done1.MaintenanceDepotOperations(a1, m1, \vec{l}, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC}))
\end{aligned}$$

$$AH_{\text{arrivedAtMaintenance}}(\text{arrivedAtMaintenance}, done1) \stackrel{def}{=} \overline{arrivedAtMaintenance}.\overline{done1}.0$$

1.8.3 WAITINGFORNETWORK STATE

Following Rules 5, 11 and 14:

$$\begin{aligned}
WaitingForNetwork(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}) \stackrel{def}{=} \\
& a1.WaitingForNetwork(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}) + \\
& evint1(z).([z = \text{retryTimerExpired}]\overline{rtc1}.RetryingMigration(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}) + \\
& \sum_{i \neq \text{retryTimerExpired}} [z = e_i]\overline{rtc1}.WaitingForNetwork(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC})) + \\
& (\nu g)\overline{timeoutExceeded}\langle g\rangle.g(y).([y = true]AbortingMission(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}) + \\
& \quad [y = false]WaitingForNetwork(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}))
\end{aligned}$$

1.8.4 RETRYINGMIGRATION STATE WITH MULTIPLE CONDITIONAL BRANCHES

Following Rule 5 and 14:

$$
\begin{aligned}
&RetryingMigration(a1, m1, \vec{l}, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}) \stackrel{def}{=} \\
&\quad a1.RetryingMigration(a1, m1, \vec{l}, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}) + \\
&\quad (\nu g1)\overline{\text{networkRestored}}\langle g1\rangle.g1(y1). \\
&\quad\quad ([y1 = true](\nu g2)\overline{\text{atMigrationStep1}}\langle g2\rangle.g2(y2). \\
&\quad\quad\quad ([y2 = true]TravelingToPolice(a1, m1, \vec{l}, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}) + \\
&\quad\quad\quad [y2 = false]RetryingMigration(a1, m1, \vec{l}, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC})) + \\
&\quad\quad [y1 = false]RetryingMigration(a1, m1, \vec{l}, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC})) + \\
&\quad (\nu g3)\overline{\text{networkRestored}}\langle g3\rangle.g3(y3). \\
&\quad\quad ([y3 = true](\nu g4)\overline{\text{atMigrationStep2}}\langle g4\rangle.g4(y4). \\
&\quad\quad\quad ([y4 = true]TravelingToEmergency(a1, m1, \vec{l}, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}) + \\
&\quad\quad\quad [y4 = false]RetryingMigration(a1, m1, \vec{l}, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC})) + \\
&\quad\quad [y3 = false]RetryingMigration(a1, m1, \vec{l}, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC})) + \\
&\quad (\nu g5)\overline{\text{networkRestored}}\langle g5\rangle.g5(y5). \\
&\quad\quad ([y5 = true](\nu g6)\overline{\text{atMigrationStep3}}\langle g6\rangle.g6(y6). \\
&\quad\quad\quad ([y6 = true]TravelingToMaintenance(a1, m1, \vec{l}, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}) + \\
&\quad\quad\quad [y6 = false]RetryingMigration(a1, m1, \vec{l}, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC})) + \\
&\quad\quad [y5 = false]RetryingMigration(a1, m1, \vec{l}, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC})) + \\
&\quad (\nu g7)\overline{\text{networkRestored}}\langle g7\rangle.g7(y7). \\
&\quad\quad ([y7 = true](\nu g8)\overline{\text{atMigrationStepReturn}}\langle g8\rangle.g8(y8). \\
&\quad\quad\quad ([y8 = true]ReturningToTMCA(a1, m1, \vec{l}, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}) + \\
&\quad\quad\quad [y8 = false]RetryingMigration(a1, m1, \vec{l}, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC})) + \\
&\quad\quad [y7 = false]RetryingMigration(a1, m1, \vec{l}, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC})) + \\
&\quad (\nu g9)\overline{\text{maxRetriesExceeded}}\langle g9\rangle.g9(y9). \\
&\quad\quad ([y9 = true]AbortingMission(a1, m1, \vec{l}, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}) + \\
&\quad\quad [y9 = false]RetryingMigration(a1, m1, \vec{l}, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}))
\end{aligned}
$$

1.8.5 ABORTINGMISSION STATE

Following Rules 5, 6 and 16:

$$
\begin{aligned}
&AbortingMission(a1, m1, \vec{l}, \vec{act}_{MIC}) \stackrel{def}{=} \\
&\quad a1.AbortingMission(a1, m1, \vec{l}, \vec{act}_{MIC}) + \\
&\quad (\nu done1)(AH_{\text{prepareFailureReport}}(\text{prepareFailureReport}, done1) \mid \\
&\quad\quad done1.ReturningToTMCA(a1, m1, \vec{l}, \vec{act}_{MIC}))
\end{aligned}
$$

where the action handler is:

$$
AH_{\text{prepareFailureReport}}(\text{prepareFailureReport}, done1) \stackrel{def}{=} \overline{prepareFailureReport}.\overline{done1}.0
$$

1.9 Non-Concurrent Composite State: PoliceStationOperations

Following Rules 22, 23, 14 and 16 the PoliceStationOperations composite state contains a sequence of substates:

$$PoliceStationOperations(a1) \stackrel{def}{=} a1.PoliceStationOperations(a1)+ \\ \text{finish}.(\nu done1)(AH_{\text{missionStep1Complete}}(\text{missionStep1Complete}, done1) \mid \\ done1.TravelingToEmergency(a1))$$

where the action handler is:

$$AH_{\text{missionStep1Complete}}(\text{missionStep1Complete}, done1) \stackrel{def}{=} \overline{missionStep1Complete}.\overline{done1}.0$$

The substates within PoliceStationOperations:

$$AuthenticatingPolice(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}) \stackrel{def}{=} \\ a1.AuthenticatingPolice(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC})+ \\ (\nu g)\overline{\text{authenticationSuccess}}\langle g\rangle.g(y). \\ ([y = true]QueryingPoliceDatabase(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC})+ \\ [y = false]AuthenticatingPolice(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}))+ \\ (\nu g)\overline{\text{authenticationFailure}}\langle g\rangle.g(y). \\ ([y = true]WaitingRetryAuth(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC})+ \\ [y = false]AuthenticatingPolice(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}))$$

$$QueryingPoliceDatabase(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC}) \stackrel{def}{=} \\ a1.QueryingPoliceDatabase(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC})+ \\ (\nu done1)(AH_{\text{queryExecuted}}(\text{queryExecuted}, done1) \mid \\ done1.CollectingIncidentData(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC}))$$

where the action handler is:

$$AH_{\text{queryExecuted}}(\text{queryExecuted}, done1) \stackrel{def}{=} \overline{queryExecuted}.\overline{done1}.0$$

$$CollectingIncidentData(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC}) \stackrel{def}{=} \\ a1.CollectingIncidentData(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC})+ \\ (\nu done2)(AH_{\text{dataRetrieved}}(\text{dataRetrieved}, done2) \mid \\ done2.PackagingPoliceData(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC}))$$

where the action handler is:

$$AH_{\text{dataRetrieved}}(\text{dataRetrieved}, done2) \stackrel{def}{=} \overline{dataRetrieved}.\overline{done2}.0$$

$$PackagingPoliceData(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC}) \stackrel{def}{=}$$
$$a1.PackagingPoliceData(a2, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC}) +$$
$$\tau.(\nu done3)(AH_{\text{packagingComplete}}(\text{packagingComplete}, done3) \mid$$
$$done3.NotifyingTMCAProgressPolice(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC}))$$

where the action handler is:

$$AH_{\text{packagingComplete}}(\text{packagingComplete}, done3) \stackrel{def}{=} \overline{packagingComplete}.\overline{done3}.0$$

$$NotifyingTMCAProgressPolice(a1, evext2, \vec{act}_{MIC}) \stackrel{def}{=}$$
$$a1.NotifyingTMCAProgressPolice(a1, evext2, \vec{act}_{MIC}) +$$
$$(\nu done4)(AH_{\text{progressUpdate}}(evext2, \text{progressUpdate}, done4) \mid done4.\overline{\text{finish}}.0)$$

where:

$$AH_{\text{progressUpdate}}(evext2, \text{progressUpdate}, done4) \stackrel{def}{=} \overline{evext2}\langle \text{progressUpdate} \rangle.\overline{done4}.0$$

The composite state behavior when entering from TravelingToPolice needs to add this fragment:

$$TravelingToPolice \stackrel{def}{=} \tau.(\nu \text{finish})(PoliceStationOperations(a1) \mid AuthenticatingPolice(a1))$$

1.10 Non-Concurrent Composite State: EmergencyServicesOperations

Similarly, following Rules 22, 23, 14 and 16 :

$$EmergencyServicesOperations(a1) \stackrel{def}{=} a1.EmergencyServicesOperations(a1) +$$
$$\text{finish}.(\nu done1)(AH_{\text{missionStep2Complete}}(\text{missionStep2Complete}, done1) \mid$$
$$done1.TravelingToMaintenance(a1))$$

where the action handler is:

$$AH_{\text{missionStep2Complete}}(\text{missionStep2Complete}, done1) \stackrel{def}{=} \overline{missionStep2Complete}.\overline{done1}.0$$

The substates follow the same pattern as PoliceStationOperations:

$$AuthenticatingEmergency(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}) \stackrel{def}{=}$$
$$a1.AuthenticatingEmergency(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}) +$$
$$(\nu g)\overline{\text{authenticationSuccess}}\langle g\rangle.g(y).$$
$$([y = true]QueryingEmergencyStatus(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC})$$
$$[y = false]AuthenticatingEmergency(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}))$$
$$(\nu g)\overline{\text{authenticationFailure}}\langle g\rangle.g(y).$$
$$([y = true]WaitingRetryAuth(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}) +$$
$$[y = false]AuthenticatingEmergency(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}))$$

$$QueryingEmergencyStatus(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC}) \stackrel{def}{=}$$
$$a1.QueryingEmergencyStatus(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC}) +$$
$$(\nu done1)(AH_{\text{queryExecuted}}(\text{queryExecuted}, done1) \mid$$
$$done1.CollectingAmbulanceData(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC}))$$

$$CollectingAmbulanceData(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC}) \stackrel{def}{=}$$
$$a1.CollectingAmbulanceData(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC}) +$$
$$(\nu done2)(AH_{\text{dataRetrieved}}(\text{dataRetrieved}, done2) \mid$$
$$done2.PackagingEmergencyData(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC}))$$

$$PackagingEmergencyData(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC}) \stackrel{def}{=}$$
$$a1.PackagingEmergencyData(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC}) +$$
$$(\nu done3)(AH_{\text{packagingComplete}}(\text{packagingComplete}, done3) \mid$$
$$done3.NotifyingTMCAProgressEmergency(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC}))$$

$$NotifyingTMCAProgressEmergency(a1, evext2, \vec{act}_{MIC}) \stackrel{def}{=}$$
$$a1.NotifyingTMCAProgressEmergency(a1, evext2, \vec{act}_{MIC}) +$$
$$(\nu done4)(AH_{\text{progressUpdate}}(evext2, \text{progressUpdate}, done4) \mid done4.\overline{\text{finish}}.0)$$

1.11 Non-Concurrent Composite State: MaintenanceDepotOperations

Following Rules 22, 23, 14 and 16 :

$$MaintenanceDepotOperations(a1) \stackrel{def}{=} a1.MaintenanceDepotOperations(a1) +$$
$$\text{finish}.(\nu done1)(AH_{\text{missionStep3Complete}}(\text{missionStep3Complete}, done1) \mid$$
$$done1.ConsolidatingAllData(a1))$$

The substates:

$$AuthenticatingMaintenance(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}) \stackrel{def}{=}$$
$$a1.AuthenticatingMaintenance(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}) +$$
$$(\nu g)\overline{\text{authenticationSuccess}}\langle g\rangle.g(y).$$
$$([y = true]QueryingEquipmentStatus(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}) +$$
$$[y = false]AuthenticatingMaintenance(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC})) +$$
$$(\nu g)\overline{\text{authenticationFailure}}\langle g\rangle.g(y).$$
$$([y = true]WaitingRetryAuth(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}) +$$
$$[y = false]AuthenticatingMaintenance(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}))$$

$$QueryingEquipmentStatus(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC}) \stackrel{def}{=}$$
$$a1.QueryingEquipmentStatus(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC}) +$$
$$(\nu done1)(AH_{\text{queryExecuted}}(\text{queryExecuted}, done1) \mid$$
$$done1.CollectingMaintenanceData(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC}))$$

$$CollectingMaintenanceData(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC}) \stackrel{def}{=}$$
$$a1.CollectingMaintenanceData(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC}) +$$
$$(\nu done2)(AH_{\text{dataRetrieved}}(\text{dataRetrieved}, done2) \mid$$
$$done2.PackagingMaintenanceData(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC}))$$

$$PackagingMaintenanceData(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC}) \stackrel{def}{=}$$
$$a1.PackagingMaintenanceData(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC}) +$$
$$(\nu done3)(AH_{\text{packagingComplete}}(\text{packagingComplete}, done3) \mid$$
$$done3.NotifyingTMCAProgressMaintenance(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC}))$$

$$NotifyingTMCAProgressMaintenance(a1, evext2, \vec{act}_{MIC}) \stackrel{def}{=}$$
$$a1.NotifyingTMCAProgressMaintenance(a1, evext2, \vec{act}_{MIC}) +$$
$$(\nu done4)(AH_{\text{progressUpdate}}(evext2, \text{progressUpdate}, done4) \mid done4.\overline{\text{finish}}.0)$$

1.12 Concurrent Composite State: ConsolidatingAllData

Following Rules 24, 25, and 14, the ConsolidatingAllData composite state contains three concurrent regions for validating data from different sources:

$$ConsolidatingAllData(a1) \stackrel{def}{=} a1.ConsolidatingAllData(a1) + \text{join.join.join.}$$
$$(\nu g)\overline{\text{allValidationsComplete}}\langle g\rangle.g(y).$$
$$([y = true]MergingResults(a1)+$$
$$[y = false]ConsolidatingAllData(a1))$$

The composite state behavior when entering from MaintenanceDepotOperations needs to add this fragment:

$$MaintenanceDepotOperations \stackrel{def}{=} \tau.(\nu\text{join})(ConsolidatingAllData(a1) \mid$$
$$CheckingPoliceCompleteness(a1) \mid CheckingEmergencyCompleteness(a1) \mid$$
$$CheckingMaintenanceCompleteness(a1))$$

1.12.1 REGION 1: VALIDATINGPOLICEDATA

$$CheckingPoliceCompleteness(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}) \stackrel{def}{=}$$
$$a1.CheckingPoliceCompleteness(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC})+$$
$$(\nu g)\overline{\text{completenessOK}}\langle g\rangle.g(y).$$
$$([y = true]CheckingPoliceConsistency(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC})+$$
$$[y = false]CheckingPoliceCompleteness(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}))$$

$$CheckingPoliceConsistency(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}) \stackrel{def}{=}$$
$$a1.CheckingPoliceConsistency(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC})+$$
$$(\nu g)\overline{\text{consistencyOK}}\langle g\rangle.g(y).$$
$$([y = true]PoliceValid(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC})+$$
$$[y = false]CheckingPoliceConsistency(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}))$$

$$PoliceValid(a1) \stackrel{def}{=} \overline{\text{join}}.0$$

1.12.2 REGION 2: VALIDATINGEMERGENCYDATA

$$CheckingEmergencyCompleteness(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}) \stackrel{def}{=}$$
$$a1.CheckingEmergencyCompleteness(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC})+$$
$$(\nu g)\overline{\text{completenessOK}}\langle g\rangle.g(y).$$
$$([y = true]CheckingEmergencyConsistency(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC})+$$
$$[y = false]CheckingEmergencyCompleteness(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}))$$

$$CheckingEmergencyConsistency(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}) \stackrel{def}{=}$$
$$a1.CheckingEmergencyConsistency(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}) +$$
$$(\nu g)\overline{\text{consistencyOK}}\langle g\rangle.g(y).$$
$$([y = true]EmergencyValid(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}) +$$
$$[y = false]CheckingEmergencyConsistency(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}))$$

$$EmergencyValid(a1) \stackrel{def}{=} \overline{\text{join}}.0$$

1.12.3 REGION 3: VALIDATINGMAINTENANCEDATA

$$CheckingMaintenanceCompleteness(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}) \stackrel{def}{=}$$
$$a1.CheckingMaintenanceCompleteness(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}) +$$
$$(\nu g)\overline{\text{completenessOK}}\langle g\rangle.g(y).$$
$$([y = true]CheckingMaintenanceConsistency(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}) +$$
$$[y = false]CheckingMaintenanceCompleteness(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}))$$

$$CheckingMaintenanceConsistency(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}) \stackrel{def}{=}$$
$$a1.CheckingMaintenanceConsistency(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}) +$$
$$(\nu g)\overline{\text{consistencyOK}}\langle g\rangle.g(y).$$
$$([y = true]MaintenanceValid(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}) +$$
$$[y = false]CheckingMaintenanceConsistency(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}))$$

$$MaintenanceValid(a1) \stackrel{def}{=} \overline{\text{join}}.0$$

1.13 Remaining Main Flow States

1.13.1 MERGINGRESULTS STATE

Following Rules 16 and 14:

$$MergingResults(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC}, \vec{c}_{MIC}) \stackrel{def}{=}$$
$$a1.MergingResults(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC}, \vec{c}_{MIC}) +$$
$$(\nu done1)(AH_{\text{mergeComplete}}(\text{mergeComplete}, done1) \mid$$
$$done1.CreatingFinalReport(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC}))$$

where the action handler is:

$$AH_{\text{mergeComplete}}(\text{mergeComplete}, done1) \stackrel{def} \overline{mergeComplete}.\overline{done1}.0$$

1.13.2 CREATINGFINALREPORT STATE

$$CreatingFinalReport(a1, m1, \vec{l}, evext2, \vec{act}_{MIC}) \stackrel{def}{=}$$
$$a1.CreatingFinalReport(a1, m1, \vec{l}, evext2, \vec{act}_{MIC}) +$$
$$(\nu done1)(AH_{\text{results}}(evext2, \text{results}, done1) \mid$$
$$done1.ReturningToTMCA(a1, m1, \vec{l}, evext2, \vec{act}_{MIC}))$$

where the action handler for remote send is:

$$AH_{\text{results}}(evext2, \text{results}, done1) \stackrel{def}{=} \overline{evext2}\langle \text{results} \rangle.\overline{done1}.0$$

1.13.3 RETURNINGTOTMCA STATE

Following Rules 8, 11, 16:

$$ReturningToTMCA(a1, m1, \vec{l}, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC}) \stackrel{def}{=}$$
$$a1.ReturningToTMCA(a1, m1, \vec{l}, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC}) +$$
$$evint1(z).([z = \text{networkFailure}]\overline{rtc1}.WaitingForNetwork(a1, m1, \vec{l}, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC}) +$$
$$\sum_{i \neq \text{networkFailure}} [z = e_i]\overline{rtc1}.ReturningToTMCA(a1, m1, \vec{l}, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC})) +$$
$$\overline{m1}\langle l1 \rangle.(\nu done1)(AH_{\text{arrivedAtTMCA}}(\text{arrivedAtTMCA}, done1) \mid$$
$$done1.ReportingToTMCA(a1, m1, \vec{l}, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC}))$$

where the action handler is:

$$AH_{\text{arrivedAtTMCA}}(\text{arrivedAtTMCA}, done1) \stackrel{def}{=} \overline{arrivedAtTMCA}.\overline{done1}.0$$

1.13.4 REPORTINGTOTMCA STATE

$$ReportingToTMCA(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC}) \stackrel{def}{=}$$
$$a1.ReportingToTMCA(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC}) +$$
$$(\nu done1)(AH_{\text{releaseResources}}(\text{releaseResources}, done1) \mid$$
$$done1.Terminated(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{act}_{MIC}))$$

where the action handler is:

$$AH_{\text{releaseResources}}(\text{releaseResources}, done1) \stackrel{def}{=} \overline{releaseResources}.\overline{done1}.0$$

1.13.5 TERMINATED STATE

$$Terminated(a1) \stackrel{def}{=} a1.Terminated(a1) + F_{MIC}(end)$$

1.14 WaitingRetryAuth State

Following Rules 11 and 14:

$$
\begin{aligned}
WaitingRetryAuth(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}) \stackrel{def}{=}\\
&a1.WaitingRetryAuth(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC})+\\
&evint1(z).([z = \text{retryTimerExpired}]\overline{rtc1}.(\nu g1)\overline{\text{atLocationPolice}}\langle g1\rangle.g1(y1).\\
&\quad([y1 = true]AuthenticatingPolice(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC})+\\
&\quad[y1 = false](\nu g2)\overline{\text{atLocationEmergency}}\langle g2\rangle.g2(y2).\\
&\quad\quad([y2 = true]AuthenticatingEmergency(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC})+\\
&\quad\quad[y2 = false](\nu g3)\overline{\text{atLocationMaintenance}}\langle g3\rangle.g3(y3).\\
&\quad\quad\quad([y3 = true]AuthenticatingMaintenance(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC})+\\
&\quad\quad\quad[y3 = false]WaitingRetryAuth(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}))))+\\
&\quad\sum_{i \neq \text{retryTimerExpired}} [z = e_i]\overline{rtc1}.WaitingRetryAuth(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}))+\\
&(\nu g)\overline{\text{maxAuthRetriesExceeded}}\langle g\rangle.g(y).\\
&\quad([y = true]AbortingMission(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC})+\\
&\quad[y = false]WaitingRetryAuth(a1, evint1, rtc1, \vec{e}_{MIC}, \vec{c}_{MIC}))
\end{aligned}
$$

1.15 Event Pool Processes

Following Rules 12 and 13, the local pool process (with maximum size 3):

$$
\begin{aligned}
A1PL_0(evint1, evext1, rtc1) &\stackrel{def}{=} evext1(x_1).A1PL_1(evint1, evext1, rtc1, x_1)\\
A1PL_1(evint1, evext1, rtc1, x_1) &\stackrel{def}{=} evext1(x_2).A1PL_2(evint1, evext1, rtc1, x_1, x_2)+\\
&\qquad \overline{evint1}\langle x_1\rangle.rtc1.A1PL_0(evint1, evext1, rtc1)\\
A1PL_2(evint1, evext1, rtc1, x_1, x_2) &\stackrel{def}{=} evext1(x_3).A1PL_3(evint1, evext1, rtc1, x_1, x_2, x_3)+\\
&\qquad \overline{evint1}\langle x_1\rangle.rtc1.A1PL_1(evint1, evext1, rtc1, x_2)\\
A1PL_3(evint1, evext1, rtc1, x_1, x_2, x_3) &\stackrel{def}{=} \overline{evint1}\langle x_1\rangle.rtc1.A1PL_2(evint1, evext1, rtc1, x_2, x_3)
\end{aligned}
$$

The remote pool process (with maximum size 2):

$$
\begin{aligned}
A1RPL_0(remote1, evext1) &\stackrel{def}{=} remote1(x_1).A1RPL_1(remote1, evext1, x_1)\\
A1RPL_1(remote1, evext1, x_1) &\stackrel{def}{=} remote1(x_2).A1RPL_2(remote1, evext1, x_1, x_2)+\\
&\qquad evext1\langle x_1\rangle.A1RPL_0(remote1, evext1)\\
A1RPL_2(remote1, evext1, x_1, x_2) &\stackrel{def}{=} evext1\langle x_1\rangle.A1RPL_1(remote1, evext1, x_2)
\end{aligned}
$$

1.16 Condition Evaluation Process

Following Rule 15, the condition evaluation process for the MIC system:

$$CE(\text{true}, \text{false}, \vec{c}_{MIC}) \stackrel{def}{=} \prod_{i=1}^{n} c_i(g).(g\langle\text{true}\rangle + g\langle\text{false}\rangle).CE(\text{true}, \text{false}, \vec{c}_{MIC})$$

where $\vec{c}_{MIC}$ includes all the condition channels defined earlier.

1.17 Agent Process

Following Rule 10, the MIC agent process is:

$$\begin{aligned} A_1(a1, m1, \vec{evext}, \vec{remote}) \stackrel{def}{=} (\nu evint1, rtc1)((\nu\vec{e}_{MIC}, \vec{c}_{MIC}, \vec{act}_{MIC})(\\ I_{MIC}(createagent)) \mid \\ A1PL_0(evint1, evext1, rtc1)) \mid A1RPL_0(remote1, evext1) \end{aligned}$$

1.18 Complete System

Following Rule 27, the complete MIC system is:

$$\begin{aligned} MIC\text{-}System(\vec{evext}, \vec{remote}) \stackrel{def}{=} (\nu a1, m1, a2, m2, a3, m3, a4, m4, a5, m5, \vec{l})(\\ A_1(a1, m1, \vec{evext}, \vec{remote}) \mid A_2(a2, m2, \vec{evext}, \vec{remote}) \mid \\ A_3(a3, m3, \vec{evext}, \vec{remote}) \mid A_4(a3, m3, \vec{evext}, \vec{remote}) \mid \\ A_5(a5, m5, \vec{evext}, \vec{remote}) \mid Pc(\vec{l}, a1, m1) \mid CE(\text{true}, \text{false}, \vec{c}_{MIC})) \end{aligned}$$

where A_2 represents the TMCA stationary agent process at place P1.

2 Pi-Calculus Formalization of other MDSs

2.1 TMCA Agent Process Definitions

2.1.1 Channel Definitions

$$\vec{e}_{TMCA} \stackrel{def}{=} \text{incidentReport}, \text{progressUpdate}, \text{results}, \text{createAgent}, \text{userAcknowledges}$$

$$\vec{c}_{TMCA} \stackrel{def}{=} \text{systemReady}, \text{allTDESARegistered}, \text{reportAnalyzed}$$

$$\vec{act}_{TMCA} \stackrel{def}{=} \text{agentCreated}, \text{dispatchCommand}, \text{resultsDisplayed}$$

2.1.2 Initial Pseudo-State

$$I_{TMCA}(start) \stackrel{def}{=} start.Initializing$$

2.1.3 STATE PROCESSES

$$Initializing(a2) \stackrel{def}{=} a2.Initializing(a2)$$

$$CreatingMIC(a2) \stackrel{def}{=} a2.CreatingMIC(a2)$$

$$DispatchingMIC(a2) \stackrel{def}{=} a2.DispatchingMIC(a2)$$

$$TrackingMIC(a2) \stackrel{def}{=} a2.TrackingMIC(a2)$$

$$ReceivingResults(a2) \stackrel{def}{=} a2.ReceivingResults(a2)$$

$$NormalMonitoring(a2) \stackrel{def}{=} a2.NormalMonitoring(a2)$$

$$ProcessingReport(a2) \stackrel{def}{=} a2.ProcessingReport(a2)$$

$$DisplayingInGUI(a2) \stackrel{def}{=} a2.DisplayingInGUI(a2)$$

2.1.4 TRANSITIONS

$$\begin{aligned} Initializing(a2, \vec{c}_{TMCA}) \stackrel{def}{=} \\ & a2.Initializing(a2, \vec{c}_{TMCA}) + (\nu g1)\overline{\text{systemReady}}\langle g1\rangle.g1(y1). \\ & ([y1 = true](\nu g2)\overline{\text{allTDESARegistered}}\langle g2\rangle.g2(y2). \\ & ([y2 = true]NormalMonitoring(a2, \vec{c}_{TMCA}) + \\ & [y2 = false]Initializing(a2, \vec{c}_{TMCA})) + \\ & [y1 = false]Initializing(a2, \vec{c}_{TMCA})) \end{aligned}$$

$$\begin{aligned} NormalMonitoring(a2, evint2, rtc2, \vec{e}_{TMCA}) \stackrel{def}{=} \\ & a2.NormalMonitoring(a2, evint2, rtc2, \vec{e}_{TMCA}) + \\ & evint2(z).([z = \text{incidentReport}]\overline{rtc2}.IncidentHandling(a2, evint2, rtc2, \vec{e}_{TMCA}) + \\ & \sum_{i \neq \text{incidentReport}} [z = e_i]\overline{rtc2}.NormalMonitoring(a2, evint2, rtc2, \vec{e}_{TMCA})) + \\ & \overline{finish}.0 \end{aligned}$$

$$ProcessingReport(a2, \vec{c}_{TMCA}) \stackrel{def}{=}$$
$$a2.ProcessingReport(a2, \vec{c}_{TMCA}) +$$
$$(\nu g1)\overline{\text{reportAnalyzed}}\langle g1\rangle.g1(y1).$$
$$([y1 = true]DisplayingInGUI(a2, \vec{c}_{TMCA}) +$$
$$[y1 = false]ProcessingReport(a2, \vec{c}_{TMCA}))$$

$$DisplayingInGUI(a2, evint2, rtc2, \vec{e}_{TMCA}) \stackrel{def}{=}$$
$$a2.DisplayingInGUI(a2, evint2, rtc2, \vec{e}_{TMCA}) +$$
$$evint2(z).([z = \text{userAcknowledges}]\overline{rtc2}.\overline{finish}.0 +$$
$$\sum_{i \neq \text{userAcknowledges}} [z = e_i]\overline{rtc2}.DisplayingInGUI(a2, evint2, rtc2, \vec{e}_{TMCA}))$$

$$CreatingMIC(a2, \vec{act}_{TMCA}) \stackrel{def}{=}$$
$$a2.CreatingMIC(a2, \vec{act}_{TMCA}) +$$
$$(\nu done1)(AH_{\text{agentCreated}}(\text{agentCreated}, done1) \mid$$
$$done1.DispatchingMIC(a2, \vec{act}_{TMCA}))$$

$$AH_{\text{agentCreated}}(\text{agentCreated}, done1) \stackrel{def}{=} \overline{\text{agentCreated}}.\overline{done1}.0$$

$$DispatchingMIC(a2, evext1, \vec{act}_{TMCA}) \stackrel{def}{=}$$
$$a2.DispatchingMIC(a2, evext1, \vec{act}_{TMCA}) +$$
$$(\nu done1)(AH_{\text{dispatchCommand}}(evext1, \text{dispatchCommand}, done1) \mid$$
$$done1.TrackingMIC(a2, evext1, \vec{act}_{TMCA}))$$

$$AH_{\text{dispatchCommand}}(evext1, \text{dispatchCommand}, done1) \stackrel{def}{=} \overline{evext1}\langle\text{dispatchCommand}\rangle.\overline{done1}.0$$

$$TrackingMIC(a2, evint2, rtc2, \vec{e}_{TMCA}) \stackrel{def}{=}$$
$$a2.TrackingMIC(a2, evint2, rtc2, \vec{e}_{TMCA}) +$$
$$evint2(z).([z = \text{progressUpdate}]\overline{rtc2}.TrackingMIC(a2, evint2, rtc2, \vec{e}_{TMCA}) +$$
$$[z = \text{results}]\overline{rtc2}.ReceivingResults(a2, evint2, rtc2, \vec{e}_{TMCA}) +$$
$$\sum_{i \notin \{\text{progressUpdate,results}\}} [z = e_i]\overline{rtc2}.TrackingMIC(a2, evint2, rtc2, \vec{e}_{TMCA}))$$

$$ReceivingResults(a2, \vec{act}_{TMCA}) \stackrel{def}{=} a2.ReceivingResults(a2, \vec{act}_{TMCA}) + (\nu done1)(AH_{\text{resultsDisplayed}}(\text{resultsDisplayed}, done1) \mid done1.NormalMonitoring(a2, \vec{act}_{TMCA}))$$

$$AH_{\text{resultsDisplayed}}(\text{resultsDisplayed}, done1) \stackrel{def}{=} \overline{\text{resultsDisplayed}}.\overline{done1}.0$$

2.1.5 COMPOSITE STATE: INCIDENTHANDLING

$$IncidentHandling(a2) \stackrel{def}{=} a2.IncidentHandling(a2) + \text{finish}.NormalMonitoring(a2)$$

$$NormalMonitoring \stackrel{def}{=} \tau.(\nu\text{finish})(IncidentHandling(a2) \mid ProcessingReport(a2))$$

2.1.6 COMPOSITE STATE: MONITORINGSYSTEM

$$MonitoringSystem(a2) \stackrel{def}{=} a2.MonitoringSystem(a2) + finish.evint2(z).([z = \text{createAgent}]\overline{rtc2}.CreatingMIC(a2, \vec{act}_{TMCA}) + \sum_{i \neq \text{createAgent}} [z = e_i]\overline{rtc2}.MonitoringSystem(a2))$$

$$Initializing \stackrel{def}{=} \tau.(\nu\text{finish})(MonitoringSystem(a2) \mid NormalMonitoring(a2))$$

2.1.7 EVENT POOL PROCESSES

$$A2PL_0(evint2, evext2, rtc2) \stackrel{def}{=} evext2(x_1).A2PL_1(evint2, evext2, rtc2, x_1)$$

$$A2PL_1(evint2, evext2, rtc2, x_1) \stackrel{def}{=} evext2(x_2).A2PL_2(evint2, evext2, rtc2, x_1, x_2) + \overline{evint2}\langle x_1\rangle.rtc2.A2PL_0(evint2, evext2, rtc2)$$

$$A2PL_2(evint2, evext2, rtc2, x_1, x_2) \stackrel{def}{=} \overline{evint2}\langle x_1\rangle.rtc2.A2PL_1(evint2, evext2, rtc2, x_2)$$

$$A2RPL_0(remote2, evext2) \stackrel{def}{=} remote2(x_1).A2RPL_1(remote2, evext2, x_1)$$

$$A2RPL_1(remote2, evext2, x_1) \stackrel{def}{=} remote2(x_2).A2RPL_2(remote2, evext2, x_1, x_2) + \overline{evext2}\langle x_1\rangle.A2RPL_0(remote2, evext2)$$

$$A2RPL_2(remote2, evext2, x_1, x_2) \stackrel{def}{=} \overline{evext2}\langle x_1\rangle.A2RPL_1(remote2, evext2, x_2)$$

2.1.8 AGENT PROCESS

$$A_2(a2, \vec{evext}, \vec{remote}) \stackrel{def}{=} (\nu evint2, rtc2)((\nu \vec{e}_{TMCA}, \vec{c}_{TMCA}, \vec{act}_{TMCA})(Initializing(a2, evint2, rtc2, \vec{e}_{TMCA}, \vec{c}_{TMCA}, \vec{act}_{TMCA})) \mid A2PL_0(evint2, evext2, rtc2)) \mid A2RPL_0(remote2, evext2)$$

2.2 LRDA Agent Process Definitions

$$\vec{e}_{LRDA} \stackrel{def}{=} \text{dataReceived, hardwareFailure, hardwareRestored}$$

$$\vec{c}_{LRDA} \stackrel{def}{=} \text{hardwareConnected, registrationConfirmedTDESA, travelTimeExceedsThreshold, parametersNormal}$$

$$\vec{act}_{LRDA} \stackrel{def}{=} \text{incidentAlert}$$

2.2.1 INITIAL PSEUDO-STATE

$$I_{LRDA}(start) \stackrel{def}{=} start.Initializing$$

2.2.2 STATE PROCESSES

$$Initializing(a3) \stackrel{def}{=} a3.Initializing(a3)$$

$$Monitoring(a3) \stackrel{def}{=} a3.Monitoring(a3)$$

$$Analyzing(a3) \stackrel{def}{=} a3.Analyzing(a3)$$

$$DetectingIncident(a3) \stackrel{def}{=} a3.DetectingIncident(a3)$$

$$Offline(a3) \stackrel{def}{=} a3.Offline(a3)$$

2.2.3 TRANSITIONS

$$
\begin{aligned}
Initializing(a3, \vec{c}_{LRDA}) \stackrel{def}{=}\ & \\
& a3.Initializing(a3, \vec{c}_{LRDA}) + (\nu g1)\overline{\text{hardwareConnected}}\langle g1\rangle.g1(y1). \\
& ([y1 = true](\nu g2)\overline{\text{registrationConfirmedTDESA}}\langle g2\rangle.g2(y2). \\
& \quad ([y2 = true]Monitoring(a3, \vec{c}_{LRDA}) + \\
& \quad [y2 = false]Initializing(a2, \vec{c}_{LRDA})) + \\
& [y1 = false]Initializing(a3, \vec{c}_{LRDA}))
\end{aligned}
$$

$$
\begin{aligned}
Monitoring(a3, evint3, rtc3, \vec{e}_{LRDA}) \stackrel{def}{=}\ & \\
& a3.Monitoring(a3, evint3, rtc3, \vec{e}_{LRDA}) + \\
& evint3(z).([z = \text{dataReceived}]\overline{rtc3}.Analyzing(a3, evint3, rtc3, \vec{e}_{LRDA}) + \\
& \quad [z = \text{hardwareFailure}]\overline{rtc3}.Offline(a3, evint3, rtc3, \vec{e}_{LRDA}) + \\
& \quad \sum_{i \notin \{\text{dataReceived,hardwareFailure}\}} [z = e_i]\overline{rtc3}.Monitoring(a3, evint3, rtc3, \vec{e}_{LRDA}))
\end{aligned}
$$

$$
\begin{aligned}
Analyzing(a3, \vec{c}_{LRDA}) \stackrel{def}{=}\ & \\
& a3.Analyzing(a3, \vec{c}_{LRDA}) + \\
& (\nu g1)\overline{\text{travelTimeExceedsThreshold}}\langle g1\rangle.g1(y1). \\
& \quad ([y1 = true]DetectingIncident(a3, \vec{c}_{LRDA}) + \\
& \quad [y1 = false](\nu g2)\overline{\text{parametersNormal}}\langle g2\rangle.g2(y2). \\
& \quad \quad ([y2 = true]Monitoring(a3, \vec{c}_{LRDA}) + \\
& \quad \quad [y2 = false]Analyzing(a3, \vec{c}_{LRDA})))
\end{aligned}
$$

$$
\begin{aligned}
DetectingIncident(a3, remote5, \vec{act}_{LRDA}) \stackrel{def}{=}\ & \\
& a3.DetectingIncident(a3, remote5, \vec{act}_{LRDA}) + \\
& (\nu done1)(AH_{\text{incidentAlert}}(remote5, \text{incidentAlert}, done1) \mid \\
& \quad done1.Monitoring(a3, remote5, \vec{act}_{LRDA}))
\end{aligned}
$$

$$
AH_{\text{incidentAlert}}(evext4, \text{incidentAlert}, done1) \stackrel{def}{=} \overline{remote5}\langle \text{incidentAlert}\rangle.\overline{done1}.0
$$

$$
\begin{aligned}
Offline(a3, evint3, rtc3, \vec{e}_{LRDA}) \stackrel{def}{=}\ & \\
& a3.Offline(a3, evint3, rtc3, \vec{e}_{LRDA}) + \\
& evint3(z).([z = \text{hardwareRestored}]\overline{rtc3}.Initializing(a3, evint3, rtc3, \vec{e}_{LRDA}) + \\
& \quad \sum_{i \neq \text{hardwareRestored}} [z = e_i]\overline{rtc3}.Offline(a3, evint3, rtc3, \vec{e}_{LRDA}))
\end{aligned}
$$

2.2.4 EVENT POOL PROCESSES

$$A3PL_0(evint3, evext3, rtc3) \stackrel{def}{=} evext3(x_1).A3PL_1(evint3, evext3, rtc3, x_1)$$

$$A3PL_1(evint3, evext3, rtc3, x_1) \stackrel{def}{=} evext3(x_2).A3PL_2(evint3, evext3, rtc3, x_1, x_2) + \overline{evint3}\langle x_1\rangle.rtc3.A3PL_0(evint3, evext3, rtc3)$$

$$A3PL_2(evint3, evext3, rtc3, x_1, x_2) \stackrel{def}{=} \overline{evint3}\langle x_1\rangle.rtc3.A3PL_1(evint3, evext3, rtc3, x_2)$$

$$A3RPL_0(remote3, evext3) \stackrel{def}{=} remote3(x_1).A3RPL_1(remote3, evext3, x_1)$$

$$A3RPL_1(remote3, evext3, x_1) \stackrel{def}{=} remote3(x_2).A3RPL_2(remote3, evext3, x_1, x_2) + \overline{evext3}\langle x_1\rangle.A3RPL_0(remote3, evext3)$$

$$A3RPL_2(remote3, evext3, x_1, x_2) \stackrel{def}{=} \overline{evext3}\langle x_1\rangle.A3RPL_1(remote3, evext3, x_2)$$

2.2.5 AGENT PROCESS

$$A_3(a3, \vec{evext}, \vec{remote}) \stackrel{def}{=} (\nu evint3, rtc3)((\nu \vec{e}_{LRDA}, \vec{c}_{LRDA}, \vec{act}_{LRDA})(Initializing(a3, evint3, rtc3, \vec{e}_{LRDA}, \vec{c}_{LRDA}, \vec{act}_{LRDA})) \mid A3PL_0(evint3, evext3, rtc3)) \mid A3RPL_0(remote3, evext3)$$

2.3 VCDA Agent Process Definitions

2.3.1 CHANNEL DEFINITIONS

$$\vec{e}_{VCDA} \stackrel{def}{=} \text{verificationRequest, cameraFailure, cameraRestored}$$

$$\vec{c}_{VCDA} \stackrel{def}{=} \text{cameraConnected, registrationConfirmedTDESA, analysisComplete}$$

$$\vec{act}_{VCDA} \stackrel{def}{=} \text{videoStreamAcquired, verificationReport}$$

2.3.2 INITIAL PSEUDO-STATE

$$I_{VCDA}(start) \stackrel{def}{=} start.Initializing$$

2.3.3 STATE PROCESSES

$$Initializing(a4) \stackrel{def}{=} a4.Initializing(a4)$$

$$Idle(a4) \stackrel{def}{=} a4.Idle(a4)$$

$$Verifying(a4) \stackrel{def}{=} a4.Verifying(a4)$$

$$AnalyzingVideo(a4) \stackrel{def}{=} a4.AnalyzingVideo(a4)$$

$$Offline(a4) \stackrel{def}{=} a4.Offline(a4)$$

2.3.4 TRANSITIONS

$$\begin{aligned}
Initializing(a4, \vec{c}_{VCDA}) \stackrel{def}{=}\\
&a4.Initializing(a4, \vec{c}_{VCDA}) + (\nu g1)\overline{\text{cameraConnected}}\langle g1\rangle.g1(y1).\\
&([y1 = true](\nu g2)\overline{\text{registrationConfirmedTDESA}}\langle g2\rangle.g2(y2).\\
&\quad([y2 = true]Idle(a4, \vec{c}_{VCDA})+\\
&\quad[y2 = false]Initializing(a2, \vec{c}_{VCDA}))+\\
&[y1 = false]Initializing(a4, \vec{c}_{VCDA}))
\end{aligned}$$

$$\begin{aligned}
Idle(a4, evint4, rtc4, \vec{e}_{VCDA}) \stackrel{def}{=}\\
&a4.Idle(a4, evint4, rtc4, \vec{e}_{VCDA})+\\
&evint4(z).([z = \text{verificationRequest}]\overline{rtc4}.Verifying(a4, evint4, rtc4, \vec{e}_{VCDA})+\\
&\quad[z = \text{cameraFailure}]\overline{rtc4}.Offline(a4, evint4, rtc4, \vec{e}_{VCDA})+\\
&\quad\sum_{i \notin \{\text{verificationRequest,cameraFailure}\}} [z = e_i]\overline{rtc4}.Idle(a4, evint4, rtc4, \vec{e}_{VCDA}))
\end{aligned}$$

$$\begin{aligned}
Verifying(a4, \vec{act}_{VCDA}) \stackrel{def}{=}\\
&a4.Verifying(a4, \vec{act}_{VCDA})+\\
&(\nu done1)(AH_{\text{videoStreamAcquired}}(\text{videoStreamAcquired}, done1) \mid\\
&\quad done1.AnalyzingVideo(a4, \vec{act}_{VCDA}))
\end{aligned}$$

$$AH_{\text{videoStreamAcquired}}(\text{videoStreamAcquired}, done1) \stackrel{def}{=} \overline{\text{videoStreamAcquired}}.\overline{done1}.0$$

$$\begin{aligned}
AnalyzingVideo(a4, remote5, \vec{c}_{VCDA}, \vec{act}_{VCDA}) \stackrel{def}{=} \\
& a4.AnalyzingVideo(a4, remote5, \vec{c}_{VCDA}, \vec{act}_{VCDA}) + \\
& (\nu g1)\overline{\text{analysisComplete}}\langle g1\rangle.g1(y1). \\
& \quad ([y1 = true](\nu done1)(AH_{\text{verificationReport}}(remote5, \text{verificationReport}, done1) \mid \\
& \quad\quad done1.Idle(a4, remote5, \vec{c}_{VCDA}, \vec{act}_{VCDA})) + \\
& \quad [y1 = false]AnalyzingVideo(a4, remote5, \vec{c}_{VCDA}, \vec{act}_{VCDA}))
\end{aligned}$$

$$AH_{\text{verificationReport}}(remote5, \text{verificationReport}, done1) \stackrel{def}{=} \overline{remote5}\langle\text{verificationReport}\rangle.\overline{done1}.0$$

$$\begin{aligned}
Offline(a4, evint4, rtc4, \vec{e}_{VCDA}) \stackrel{def}{=} \\
& a4.Offline(a4, evint4, rtc4, \vec{e}_{VCDA}) + \\
& evint4(z).([z = \text{cameraRestored}]rtc4.Initializing(a4, evint4, rtc4, \vec{e}_{VCDA}) + \\
& \quad \sum_{i \neq \text{cameraRestored}} [z = e_i]\overline{rtc4}.Offline(a4, evint4, rtc4, \vec{e}_{VCDA}))
\end{aligned}$$

2.3.5 EVENT POOL PROCESSES

$$A4PL_0(evint4, evext4, rtc4) \stackrel{def}{=} evext4(x_1).A4PL_1(evint4, evext4, rtc4, x_1)$$

$$\begin{aligned}
A4PL_1(evint4, evext4, rtc4, x_1) \stackrel{def}{=} & \; evext4(x_2).A4PL_2(evint4, evext4, rtc4, x_1, x_2) + \\
& \overline{evint4}\langle x_1\rangle.rtc4.A4PL_0(evint4, evext4, rtc4)
\end{aligned}$$

$$A4PL_2(evint4, evext4, rtc4, x_1, x_2) \stackrel{def}{=} \overline{evint4}\langle x_1\rangle.rtc4.A4PL_1(evint4, evext4, rtc4, x_2)$$

$$A4RPL_0(remote4, evext4) \stackrel{def}{=} remote4(x_1).A4RPL_1(remote4, evext4, x_1)$$

$$\begin{aligned}
A4RPL_1(remote4, evext4, x_1) \stackrel{def}{=} & \; remote4(x_2).A4RPL_2(remote4, evext4, x_1, x_2) + \\
& \overline{evext4}\langle x_1\rangle.A4RPL_0(remote4, evext4)
\end{aligned}$$

$$A4RPL_2(remote4, evext4, x_1, x_2) \stackrel{def}{=} \overline{evext4}\langle x_1\rangle.A4RPL_1(remote4, evext4, x_2)$$

2.3.6 AGENT PROCESS

$$A_4(a4, \vec{evext}, \vec{remote}) \stackrel{def}{=} (\nu evint4, rtc4)((\nu \vec{e}_{VCDA}, \vec{c}_{VCDA}, \vec{act}_{VCDA})(\\ Initializing(a4, evint4, rtc4, \vec{e}_{VCDA}, \vec{c}_{VCDA}, \vec{act}_{VCDA})) \mid \\ A4PL_0(evint4, evext4, rtc4)) \mid A4RPL_0(remote4, evext4)$$

2.4 TDESA Agent Process Definitions

2.4.1 CHANNEL DEFINITIONS

$$\vec{e}_{TDESA} \stackrel{def}{=} \text{incidentAlert, verificationReport, userAcknowledges}$$

$$\vec{c}_{TDESA} \stackrel{def}{=} \text{agentReady, minimumAgentsRegistered, incidentConfirmed, incidentNotConfirmed, reportAnalyzed}$$

$$\vec{act}_{TDESA} \stackrel{def}{=} \text{verificationRequest, incidentReport}$$

2.4.2 INITIAL PSEUDO-STATE

$$I_{TDESA}(start) \stackrel{def}{=} start.Initializing$$

2.4.3 STATE PROCESSES

$$Initializing(a5) \stackrel{def}{=} a5.Initializing(a5)$$

$$NormalOperation(a5) \stackrel{def}{=} a5.NormalOperation(a5)$$

$$ExecutingTask(a5) \stackrel{def}{=} a5.ExecutingTask(a5)$$

$$CoordinatingVerification(a5) \stackrel{def}{=} a5.CoordinatingVerification(a5)$$

$$WaitingVerification(a5) \stackrel{def}{=} a5.WaitingVerification(a5)$$

$$AnalyzingReport(a5) \stackrel{def}{=} a5.AnalyzingReport(a5)$$

$$SendingReport(a5) \stackrel{def}{=} a5.SendingReport(a5)$$

$$ProcessingReport(a5) \stackrel{def}{=} a5.ProcessingReport(a5)$$

$$DisplayingInGUI(a5) \stackrel{def}{=} a5.DisplayingInGUI(a5)$$

2.4.4 TRANSITIONS

$$\begin{aligned}
Initializing(a5, \vec{c}_{TDESA}) \stackrel{def}{=} \\
& a5.Initializing(a5, \vec{c}_{TDESA}) + (\nu g1)\overline{\text{agentReady}}\langle g1\rangle.g1(y1). \\
& ([y1 = true](\nu g2)\overline{\text{minimumAgentsRegistered}}\langle g2\rangle.g2(y2). \\
& ([y2 = true]NormalOperation(a5, \vec{c}_{TDESA}) + \\
& [y2 = false]Initializing(a5, \vec{c}_{TDESA})) + \\
& [y1 = false]Initializing(a5, \vec{c}_{TDESA}))
\end{aligned}$$

$$\begin{aligned}
NormalOperation(a5, evint5, rtc5, \vec{e}_{TDESA}) \stackrel{def}{=} \\
& a5.NormalOperation(a5, evint5, rtc5, \vec{e}_{TDESA}) + \\
& evint5(z).([z = \text{incidentAlert}]\overline{rtc5}.IncidentProcessing(a5, evint5, rtc5, \vec{e}_{TDESA}) + \\
& \sum_{i \neq \text{incidentAlert}} [z = e_i]\overline{rtc5}.NormalOperation(a5, evint5, rtc5, \vec{e}_{TDESA})) + \\
& \overline{finish}.0
\end{aligned}$$

$$\begin{aligned}
CoordinatingVerification(a5, remote4, \vec{act}_{TDESA}) \stackrel{def}{=} \\
& a5.CoordinatingVerification(a5, remote4, \vec{act}_{TDESA}) + \\
& (\nu done1)(AH_{\text{verificationRequest}}(remote4, \text{verificationRequest}, done1) \mid \\
& done1.WaitingVerification(a5, remote4, \vec{act}_{TDESA}))
\end{aligned}$$

$$AH_{\text{verificationRequest}}(remote4, \text{verificationRequest}, done1) \stackrel{def}{=} \overline{remote4}\langle\text{verificationRequest}\rangle.\overline{done1}.0$$

$$WaitingVerification(a5, evint5, rtc5, \vec{e}_{TDESA}) \stackrel{def}{=}$$
$$a5.WaitingVerification(a5, evint5, rtc5, \vec{e}_{TDESA}) +$$
$$evint5(z).([z = \text{verificationReport}]\overline{rtc5}.AnalyzingReport(a5, evint5, rtc5, \vec{e}_{TDESA}) +$$
$$\sum_{i \neq \text{verificationReport}} [z = e_i]\overline{rtc5}.WaitingVerification(a5, evint5, rtc5, \vec{e}_{TDESA}))$$

$$AnalyzingReport(a5, evext2, \vec{c}_{TDESA}, \vec{act}_{TDESA}) \stackrel{def}{=}$$
$$a5.AnalyzingReport(a5, evext2, \vec{c}_{TDESA}, \vec{act}_{TDESA}) +$$
$$(\nu g1)\overline{\text{incidentConfirmed}}\langle g1\rangle.g1(y1).$$
$$([y1 = true]SendingReport(a5, evext2, \vec{c}_{TDESA}, \vec{act}_{TDESA}) +$$
$$[y1 = false](\nu g2)\overline{\text{incidentNotConfirmed}}\langle g2\rangle.g2(y2).$$
$$([y2 = true]\overline{finish}.0 +$$
$$[y2 = false]AnalyzingReport(a5, evext2, \vec{c}_{TDESA}, \vec{act}_{TDESA})))$$

$$SendingReport(a5, remote2, \vec{act}_{TDESA}) \stackrel{def}{=}$$
$$a5.SendingReport(a5, remote2, \vec{act}_{TDESA}) +$$
$$(\nu done1)(AH_{\text{incidentReport}}(remote2, \text{incidentReport}, done1) \mid$$
$$done1.NormalOperation(a5, remote2, \vec{act}_{TDESA}))$$

$$AH_{\text{incidentReport}}(remote2, \text{incidentReport}, done1) \stackrel{def}{=} \overline{remote2}\langle\text{incidentReport}\rangle.\overline{done1}.0$$

2.4.5 COMPOSITE STATE: INCIDENTPROCESSING

$$IncidentProcessing(a5) \stackrel{def}{=} a5.IncidentProcessing(a5) + \text{finish}.NormalOperation(a5)$$

$$NormalOperation \stackrel{def}{=} \tau.(\nu\text{finish})(IncidentProcessing(a5) \mid CoordinatingVerification(a5))$$

2.4.6 COMPOSITE STATE: MONITORING

$$Monitoring(a5) \stackrel{def}{=} a5.Monitoring(a5) + \text{finish}.0$$

$$Initializing \stackrel{def}{=} \tau.(\nu\text{finish})(Monitoring(a5) \mid NormalOperation(a5))$$

2.4.7 EVENT POOL PROCESSES

$$A5PL_0(evint5, evext5, rtc5) \stackrel{def}{=} evext5(x_1).A5PL_1(evint5, evext5, rtc5, x_1)$$

$$A5PL_1(evint5, evext5, rtc5, x_1) \stackrel{def}{=} evext5(x_2).A5PL_2(evint5, evext5, rtc5, x_1, x_2) + \overline{evint5}\langle x_1\rangle.rtc5.A5PL_0(evint5, evext5, rtc5)$$

$$A5PL_2(evint5, evext5, rtc5, x_1, x_2) \stackrel{def}{=} \overline{evint5}\langle x_1\rangle.rtc5.A5PL_1(evint5, evext5, rtc5, x_2)$$

$$A5RPL_0(remote5, evext5) \stackrel{def}{=} remote5(x_1).A5RPL_1(remote5, evext5, x_1)$$

$$A5RPL_1(remote5, evext5, x_1) \stackrel{def}{=} remote5(x_2).A5RPL_2(remote5, evext5, x_1, x_2) + \overline{evext5}\langle x_1\rangle.A5RPL_0(remote5, evext5)$$

$$A5RPL_2(remote5, evext5, x_1, x_2) \stackrel{def}{=} \overline{evext5}\langle x_1\rangle.A5RPL_1(remote5, evext5, x_2)$$

2.4.8 AGENT PROCESS

$$A_5(a5, \vec{evext}, \vec{remote}) \stackrel{def}{=} (\nu evint5, rtc5)((\nu \vec{e}_{TDESA}, \vec{c}_{TDESA}, \vec{act}_{TDESA})(Initializing(a5, evint5, rtc5, \vec{e}_{TDESA}, \vec{c}_{TDESA}, \vec{act}_{TDESA})) \mid A5PL_0(evint5, evext5, rtc5)) \mid A5RPL_0(remote5, evext5)$$