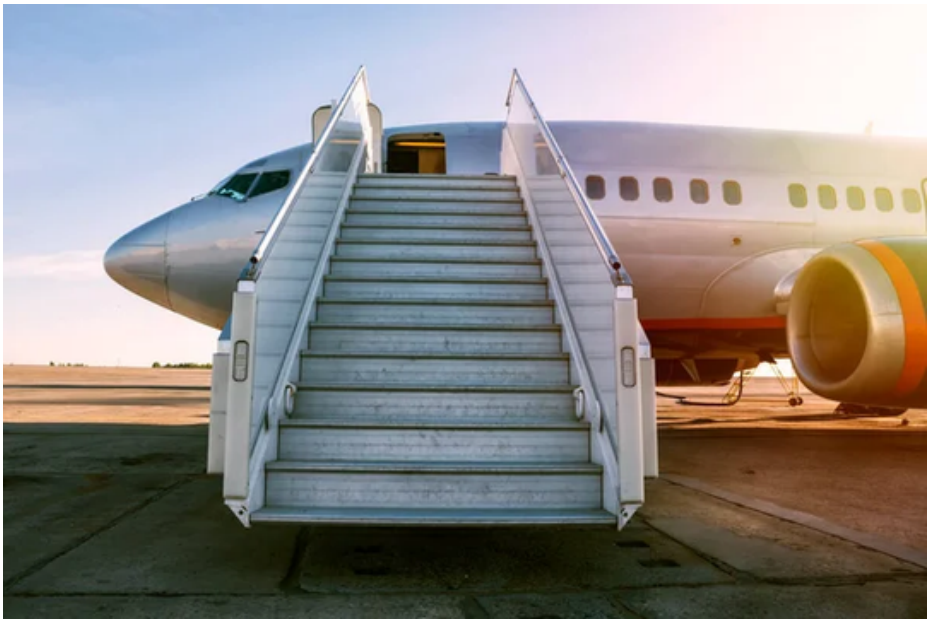
SIMULADOR A MIDA

Simulació

Quatrimestre Tardor 2022/2023



Escales de desembarcament / embarcament

Izan Cordobilla Blanco

Professor: Mnty

09/01/2023

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

FIB

INDEX

1. OBJECTE

1.1. Descripció

Les escales d'embarcament són les plataformes que es connecten a l'avió i que permeten als passatgers pujar a bord. Hi ha diversos tipus d'escales d'embarcament, com ara escales fixes, escales amb trona o escales mòbils.
Les escales de desembarcament són les que es fan servir per a que els passatgers puguin baixar de l'avió una vegada han arribat a la seva destinació. Aquestes escales solen ser similars a les escales d'embarcament, amb la diferència que es poden ajustar a diferents alçades per adaptar-se a la porta de l'avió.

1.2. Hipòtesis simplificadores

Per a una més simple realització de la pràctica, s'han tingut en compte una sèrie d'hipòtesis simplificadores:

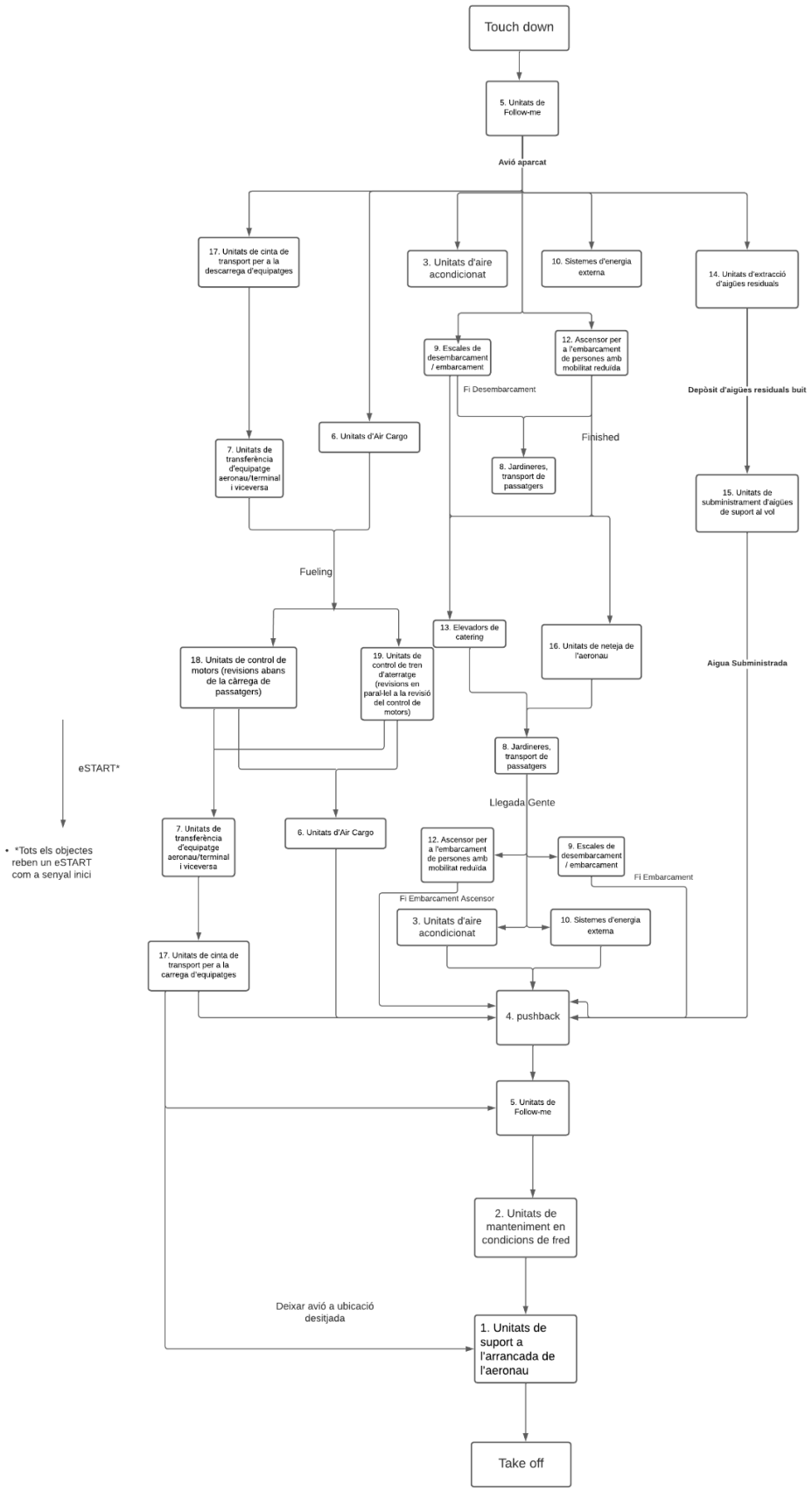
- Tots els passatgers que pugen / baixen d'un l'avió ho fan a la mateixa velocitat. Per tant, si per exemple hi ha 50 passatgers en un vol, cadascun d'aquests pujarà o baixarà en el mateix temps que un altre passatger del mateix vol.
- No s'ha tingut en compte l'alçada de l'avió per establir el temps que l'escala tarda en acoblar-se / desacoblar-se.
- No s'ha tingut en compte que hi pugui haver un accident mentre algun passatger puja o baixa les escales.
- No s'ha tingut en compte que es puguin formar cues perquè els passatgers que entren a l'avió no s'acomoden ràpidament.

1.3. Hipòtesis estructurals / dades

- El nombre de passatgers que pugen o baixen d'un avió és configurable, però es recomana que no superi els 500 passatgers (no seria realista).
- Per establir els temps que tarda l'objecte en realitzar diferents tasques, s'ha optat per una distribució triangular perquè es pot indicar el mínim, el màxim i el *peak*. A continuació es mostren els temps de les diferents tasques:
    - **L'objecte es desplaça cap a on hagi a realitzar l'event:** min: 80.0, peak: 180.0, max: 250.0.
    - **L'objecte rep una indicació de que pot començar a treballar:** min: 80.0, peak: 180.0, max: 250.0.
    - **L'objecte es situa on ha de realitzar l'event:** min: 80.0, peak: 180.0, max: 250.0.

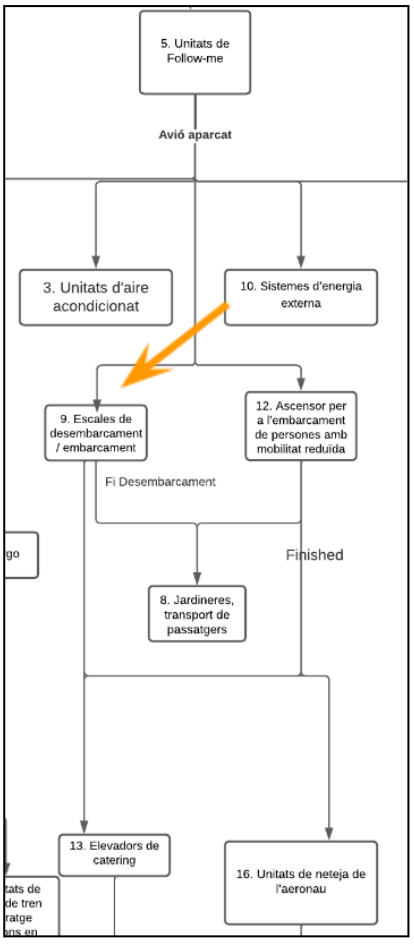
- **L’objecte s’acobla a l’avió:** minTimeAcoplant, minTimeAcoplant + 10.5, minTimeAcoplant + 20.0.
- **Totes les persones pugen / baixen a l’avió:** passatgersEmbDesemb * (min: 5.0, peak: 10.0, max: 15.0)
- **L’objecte es desacobla a l’avió:** minTimeDescoplant, minTimeDescoplant + 10.5, minTimeDescoplant + 20.0.

2. ESPECIFICACIÓ

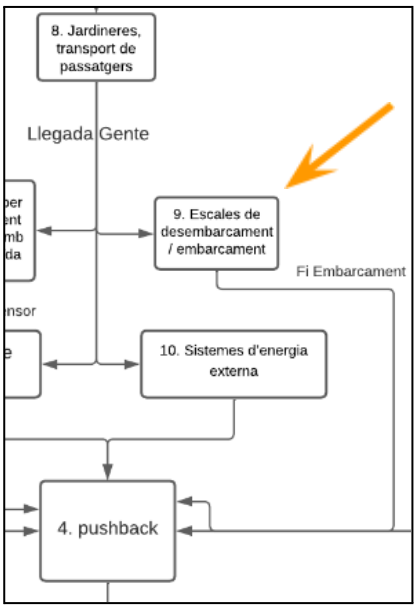


2.1. Diagrama de components

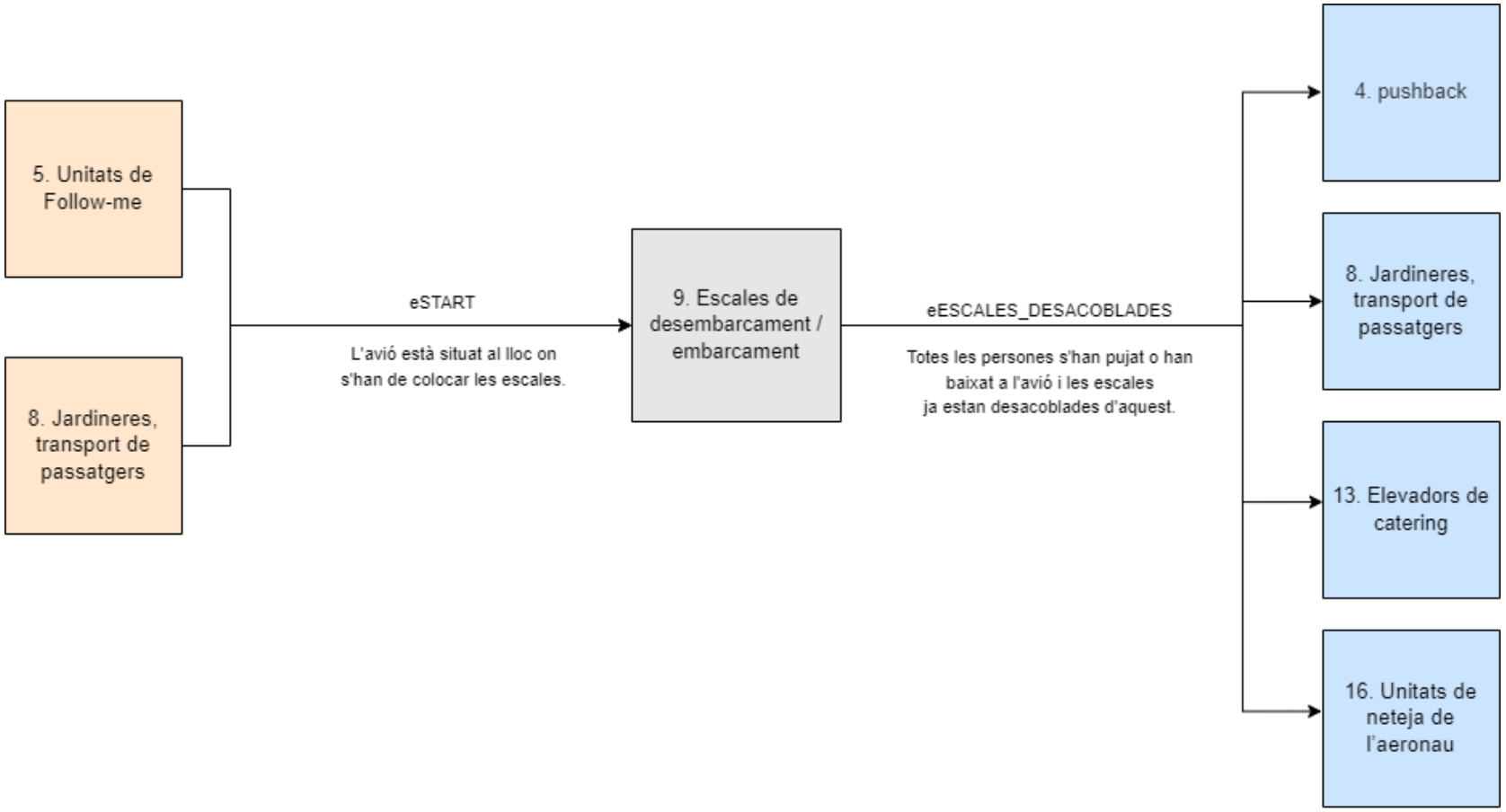
Detall del diagrama general per les escales de desembarcament:



Detall del diagrama general per les escales d'embarcament:

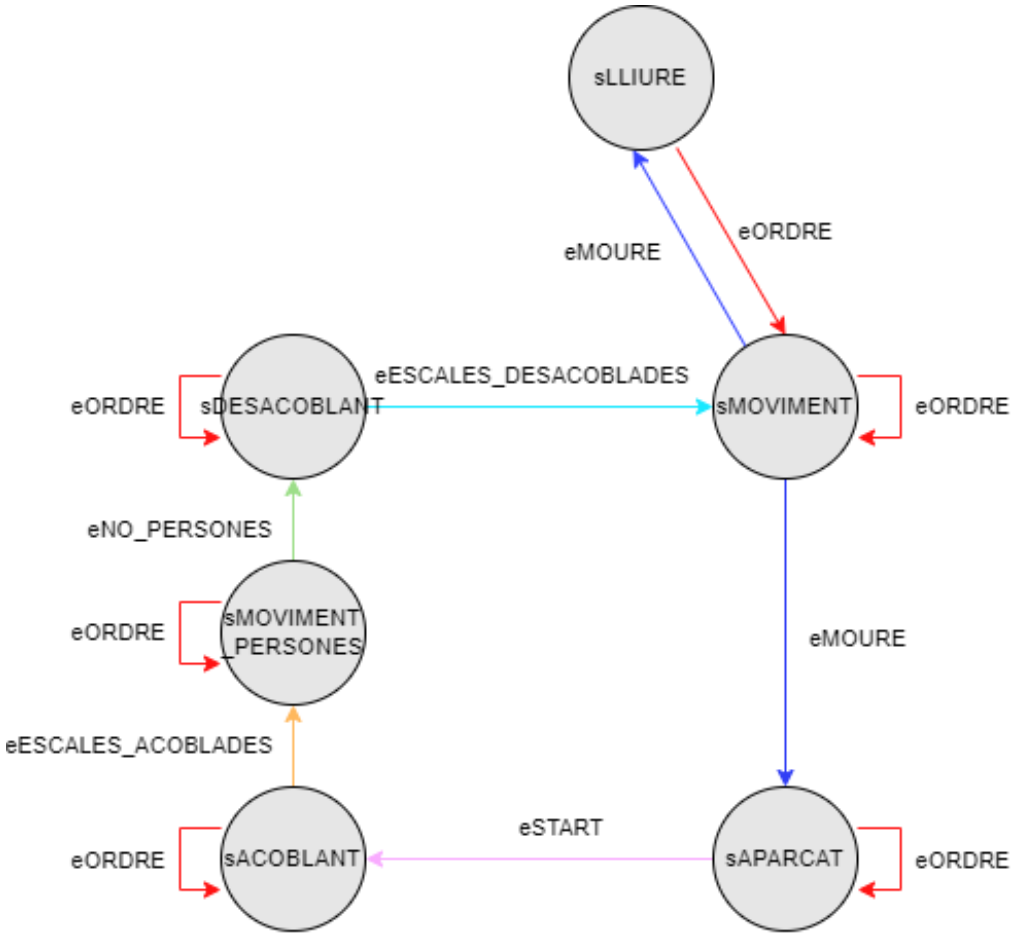


A partir dels anteriors diagrames, s'ha construït el següent diagrama de components

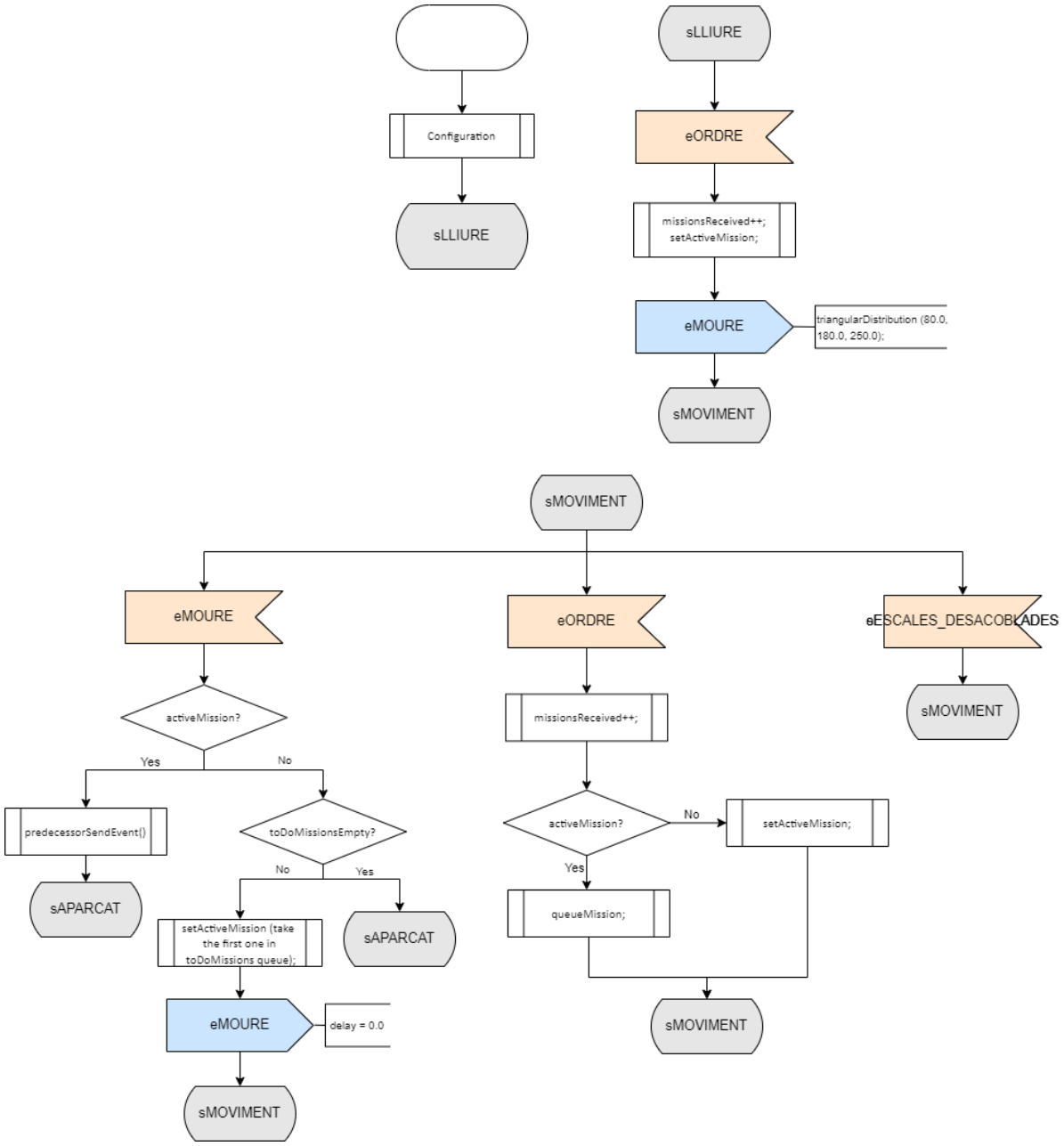


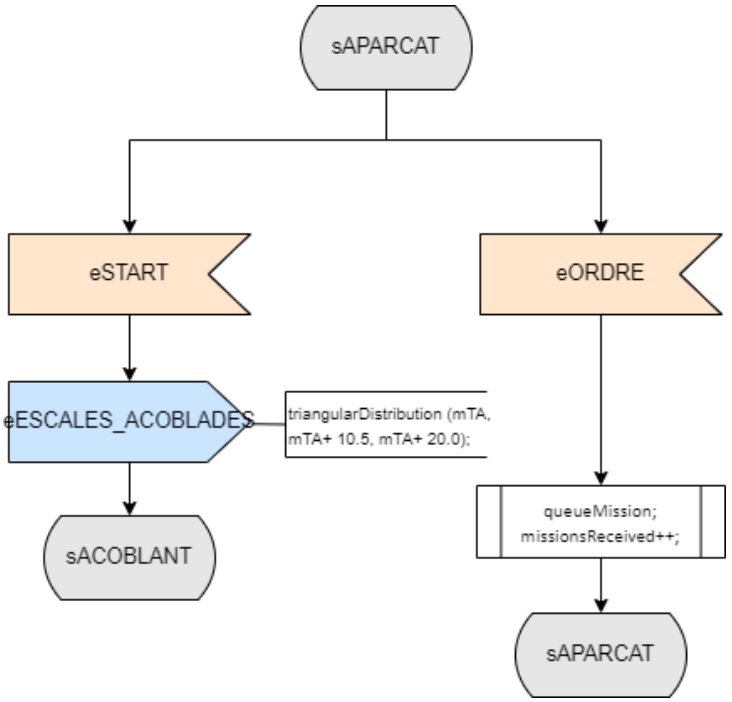
2.2. Diagrama d'estats

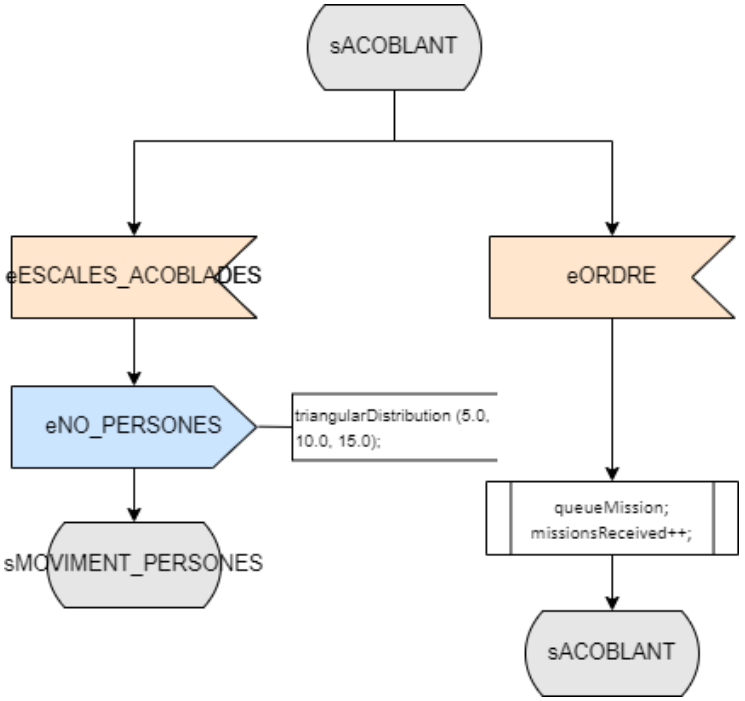
A continuació es mostra el diagrama d'estats de les escales d'embarcament / desembarcament:

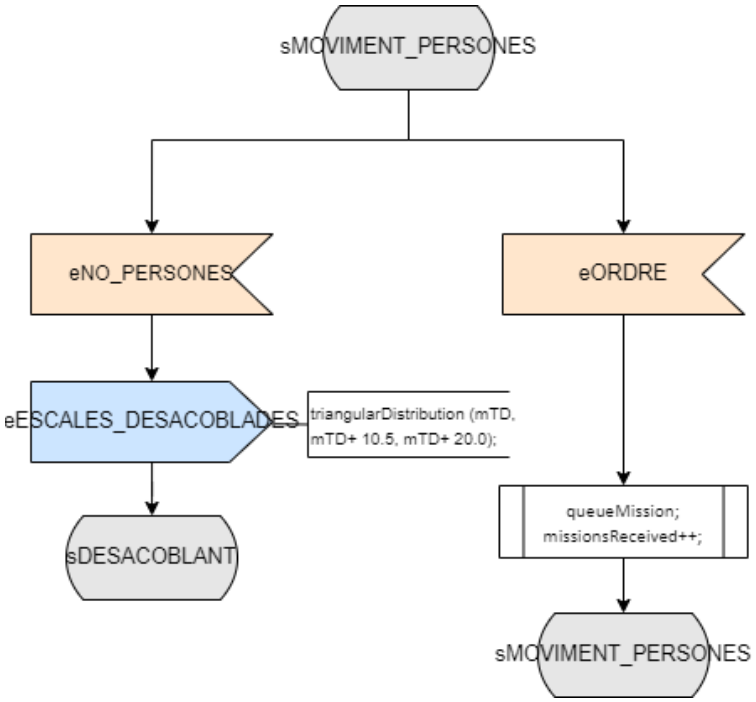


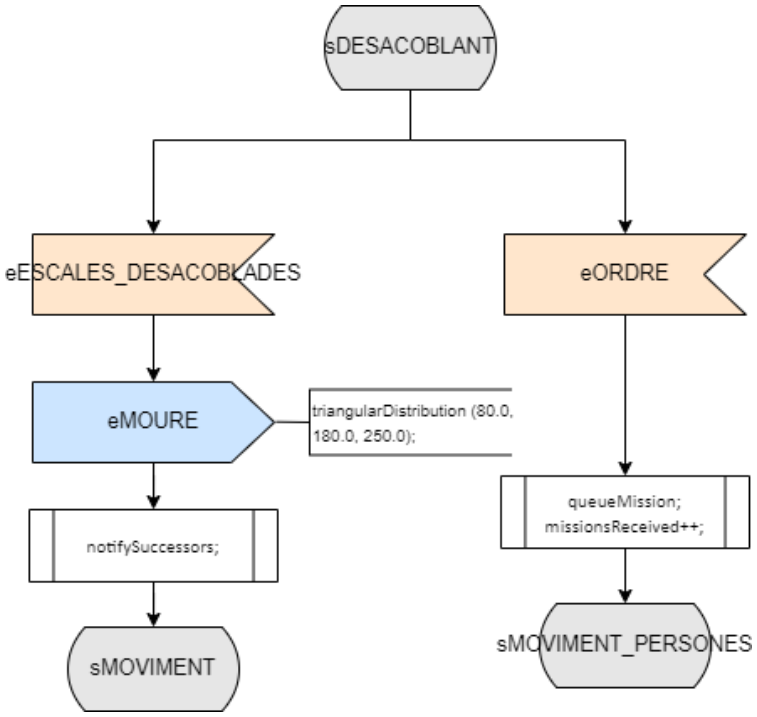
2.3. Diagrama de processos




sAPARCAT
eSTART
eORDRE
eESCALES_ACOBLADES
triangularDistribution (mTA, mTA+ 10.5, mTA+ 20.0);
queueMission; missionsReceived++;
sACOBLANT
sAPARCAT


sACOBLANT
eESCALES_ACOBLADES
eORDRE
eNO_PERSONES
triangularDistribution (5.0, 10.0, 15.0);
queueMission; missionsReceived++;
sMOVIMENT_PERSONES
sACOBLANT


sMOVIMENT_PERSONES
eNO_PERSONES
eORDRE
eESCALES_DESACOBLADES
triangularDistribution (mTD, mTD+ 10.5, mTD+ 20.0);
queueMission; missionsReceived++;
sDESACOBLANT
sMOVIMENT_PERSONES


sDESACOBLANT
eESCALES_DESACOBLADES
eORDRE
eMOURE
triangularDistribution (80.0, 180.0, 250.0);
queueMission; missionsReceived++;
notifySuccessors;
sMOVIMENT
sMOVIMENT_PERSONES

3. CODI

```
//Metode que genera un numero a l'atzar utilitzant una distribucio triangular
float CComissariObject::triangularDistribution(double min, double peak, double max)
{
    std::random_device rd;
    std::mt19937 gen(rd());

    std::array<double, 3> i{ min, peak, max };
    std::array<double, 3> w{ 0, 1, 0 };
    std::piecewise_linear_distribution<double> td =
std::piecewise_linear_distribution<double>{i.begin(), i.end(), w.begin()};

    return (float) td(gen);
}

// Metode que actualitza els estadistics
void CComissariObject::updateStatistics(float taskTime) {
    totalTime += taskTime;
    avgTimePerMission = totalTime / (float) receivedMissions;
    totalPassengers = passatgersEmbDesemb * receivedMissions;
}

//Processar un esdeveniment de simulació, funció pura que us toca implementar
void CComissariObject::processEvent (CSimulationEvent* event)  {
    switch (this->getState())
    {
        case sLLIURE: {
            if(event->getEventType() == eORDRE) {
                activeMission = event->getMission();
                receivedMissions++;
                float taskTime = triangularDistribution(80.0, 180.0, 250.0);
                updateStatistics(taskTime);

                CSimulationEvent* moureEvent = new
CSimulationEvent(m_Simulator->getCurrentTime() + taskTime, this, this, activeMission,
eMOURE);
                m_Simulator->scheduleEvent(moureEvent);
                cout << "Time: " << m_Simulator->getCurrentTime() << " --> sLLIURE: arriba
eORDRE i agafo missio. Canvi d'estat a sMOVIMENT." << endl;
                setState(sMOVIMENT);
            }
            break;
        }

        case sMOVIMENT: {
            switch (event->getEventType())
            {
```

```
            case eORDRE: {
                receivedMissions++;
                if (activeMission != NULL) {
                    toDoMissions.push(event->getMission());
                    cout << "Time: " << m_Simulator->getCurrentTime() << " --> sMOVIMENT:
arriba eORDRE pero ja tinc activeMission. Em guardo la missio." << endl;
                }

                else {
                    this->activeMission = event->getMission();
                    cout << "Time: " << m_Simulator->getCurrentTime() << " --> sMOVIMENT:
arriba eORDRE i no tinc activeMission. Comenca missio." << endl;

                }
                break;
            }

            case eMOURE: {
                if (this->activeMission == NULL) {
                    if (toDoMissions.empty()) {
                        setState(sLLIURE);
                        cout << "Time: " << m_Simulator->getCurrentTime() << " --> sMOVIMENT:
no tinc activeMission i m'arriba eMOURE però no queden missions a fer. Canvi d'estat a
sLLIURE." << endl;
                    }
                    else {
                        this->activeMission = toDoMissions.front();
                        toDoMissions.pop();
                        CSimulationEvent* startEvent = new
CSimulationEvent(m_Simulator->getCurrentTime(), this, this, this->activeMission,
eMOURE);
                        m_Simulator->scheduleEvent(startEvent);
                        cout << "Time: " << m_Simulator->getCurrentTime() << " --> sMOVIMENT:
no tinc activeMission i m'arriba eMOURE. Hi ha missions a fer i agafo una." << endl;
                    }
                }

                else {
                    m_predecessor1->sendMeEvent(new
CSimulationEvent(m_Simulator->getCurrentTime() + 10.0, m_predecessor1, this,
event->getMission(), eSTART));
                    setState(sAPARCAT);
                    cout << "Time: " << m_Simulator->getCurrentTime() << " --> sMOVIMENT:
tinc missio activa i m'arriba eMOURE. Canvi d'estat a sAPARCAT." << endl;
                }
                break;
            }
        }
```

```
            break;
        }

        case sAPARCAT: {
            switch (event->getEventType()) {
                case eSTART: {
                    float taskTime = triangularDistribution(minTimeAcoplant, minTimeAcoplant +
10.5, minTimeAcoplant + 20.0);
                    updateStatistics(taskTime);
                    CSimulationEvent* escalesAcobladesEvent = new
CSimulationEvent(m_Simulator->getCurrentTime() + taskTime, this, this,
event->getMission(), eESCALES_ACOBLADES);
                    m_Simulator->scheduleEvent(escalesAcobladesEvent);
                    setState(sACOBLANT);
                    cout << "Time: " << m_Simulator->getCurrentTime() << " --> sAPARCAT: arriba
eSTART i comenco a acoblar escales. Canvi d'estat a sACOBLANT" << endl;
                    break;
                }

                case eORDRE: {
                    toDoMissions.push(event->getMission());
                    receivedMissions++;
                    cout << "Time: " << m_Simulator->getCurrentTime() << " --> sAPARCAT: arriba
eORDRE i em guardo la missio." << endl;
                    break;
                }
            }
            break;
        }

        case sACOBLANT: {
            switch (event->getEventType()) {
                case eESCALES_ACOBLADES: {
                    float taskTimePerPerson = triangularDistribution(5.0, 10.0, 15.0);
                    float taskTime = taskTimePerPerson * passatgersEmbDesemb;
                    updateStatistics(taskTime);
                    CSimulationEvent* noPersonesEvent = new
CSimulationEvent(m_Simulator->getCurrentTime() + taskTime, this, this,
event->getMission(), eNO_PERSONES);
                    m_Simulator->scheduleEvent(noPersonesEvent);
                    setState(sMOVIMENT_PERSONES);
                    cout << "Time: " << m_Simulator->getCurrentTime() << " --> sACOBLANT:
arriba eESCALES_ACOBLADES i " << passatgersEmbDesemb << " persones comencen a
pujar o baixar. Canvi d'estat a sMOVIMENT_PERSONES" << endl;
                    break;
                }

                case eORDRE: {
```

```
                toDoMissions.push(event->getMission());
                receivedMissions++;
                cout << "Time: " << m_Simulator->getCurrentTime() << " --> sACOBLANT:
arriba eORDRE i em guardo la missio." << endl;
                break;
            }
        }
        break;
    }

    case sMOVIMENT_PERSONES: {
        switch (event->getEventType()) {
            case eNO_PERSONES: {
                float taskTime = triangularDistribution(minTimeDescoplant, minTimeDescoplant
+ 10.5, minTimeDescoplant + 20.0);
                updateStatistics(taskTime);
                CSimulationEvent* escalesDesacobladesEvent = new
CSimulationEvent(m_Simulator->getCurrentTime() + taskTime, this, this,
event->getMission(), eESCALES_DESACOBLADES);
                m_Simulator->scheduleEvent(escalesDesacobladesEvent);
                setState(sDESACOBLANT);
                cout << "Time: " << m_Simulator->getCurrentTime() << " -->
sMOVIMENT_PERSONES: arriba eNO_PERSONES i les escales es comencen a
desacoblar. Canvi d'estat a sDESACOBLANT" << endl;
                break;
            }

            case eORDRE: {
                toDoMissions.push(event->getMission());
                receivedMissions++;
                cout << "Time: " << m_Simulator->getCurrentTime() << " -->
sMOVIMENT_PERSONES: arriba eORDRE i em guardo la missio." << endl;
                break;
            }
        }
        break;
    }

    case sDESACOBLANT: {
        switch (event->getEventType()) {
            case eESCALES_DESACOBLADES: {
                float taskTime = triangularDistribution(80.0, 180.0, 250.0);
                updateStatistics(taskTime);
                CSimulationEvent* moureEvent = new
CSimulationEvent(m_Simulator->getCurrentTime() + taskTime, this, this, activeMission,
eMOURE);
                m_Simulator->scheduleEvent(moureEvent);
```

```
                // Avisar a successors que ja he acabat
                CSimulationEvent* startEvent1 = new
CSimulationEvent(m_Simulator->getCurrentTime() + taskTime, this, m_successor1,
activeMission, eSTART);
                m_Simulator->scheduleEvent(startEvent1);
                CSimulationEvent* startEvent2 = new
CSimulationEvent(m_Simulator->getCurrentTime() + taskTime, this, m_successor2,
activeMission, eSTART);
                m_Simulator->scheduleEvent(startEvent2);
                CSimulationEvent* startEvent3 = new
CSimulationEvent(m_Simulator->getCurrentTime() + taskTime, this, m_successor3,
activeMission, eSTART);
                m_Simulator->scheduleEvent(startEvent3);
                CSimulationEvent* startEvent4 = new
CSimulationEvent(m_Simulator->getCurrentTime() + taskTime, this, m_successor4,
activeMission, eSTART);
                m_Simulator->scheduleEvent(startEvent4);


                this->activeMission = NULL;
                setState(sMOVIMENT);
                cout << "Time: " << m_Simulator->getCurrentTime() << " --> sDESACOBLANT:
arriba eESCALES_DESACOBLADES. Canvi d'estat a sMOVIMENT" << endl;
                break;
            }

            case eORDRE: {
                toDoMissions.push(event->getMission());
                receivedMissions++;
                setState(sDESACOBLANT);
                cout << "Time: " << m_Simulator->getCurrentTime() << " --> sDESACOBLANT:
arriba eORDRE i em guardo la missio." << endl;
                break;
            }
        }
        break;
    }
  }
}
```