Modelarea Proceselor din Cadrul unei Companii

Nistor Marian-Sergiu
Sescu Alexandra

Table of Contents

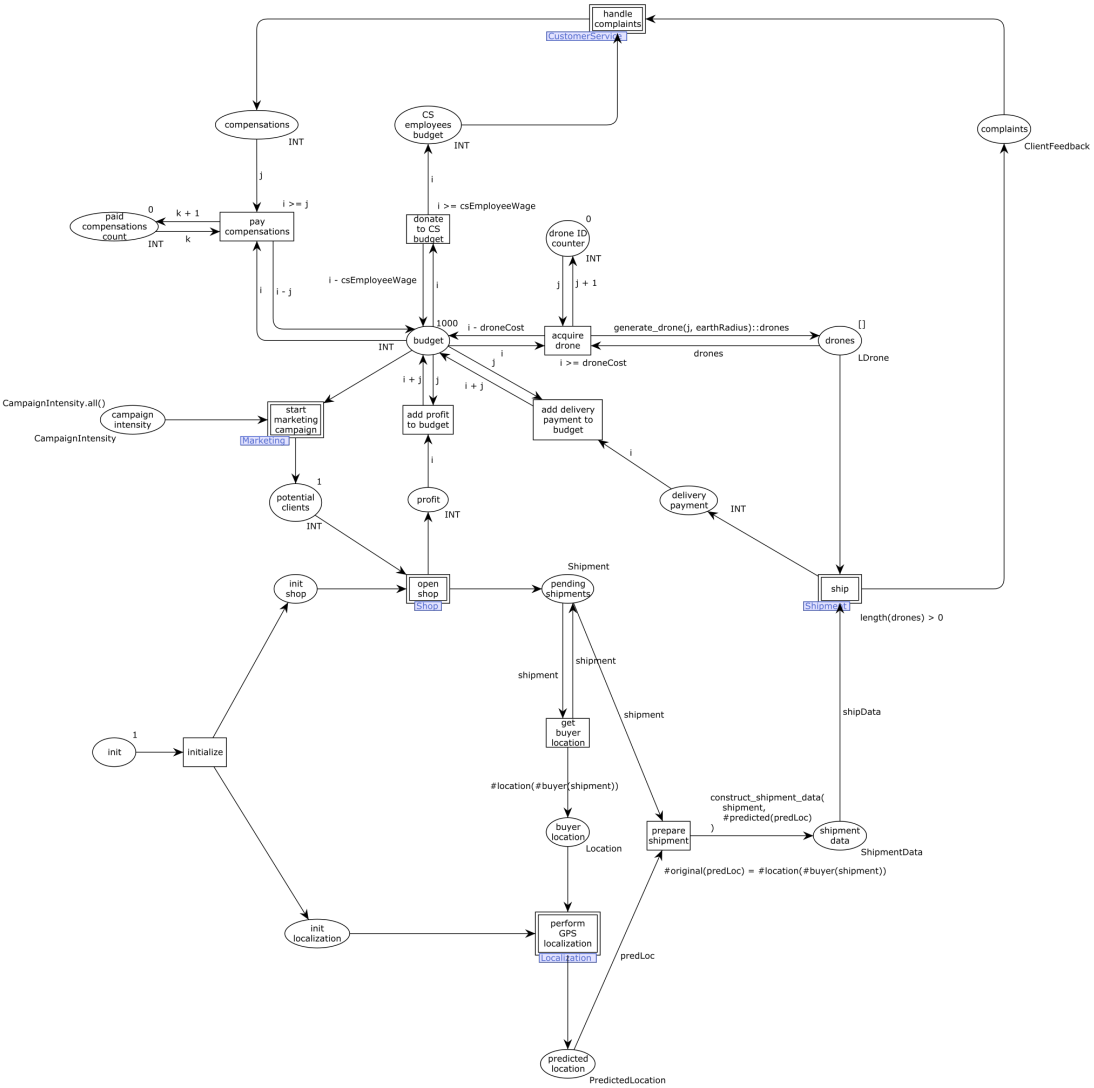
Chapter 1

Specificatiile proiectului

Obiectivul proiectului consta in modelarea proceselor din cadrul unei companii bazate pe mai multe module:

- Modulul ”HQ”, care coordoneaza departamentele
- Modulul ”Shop”, in cadrul caruia se genereaza si se comercializeaza produse
- Modulul ”Localization”, care trateaza localizarea clientilor, pe baza informatiei primite de la sateliti de GPS, in scopul transportului produsului cerut
- Modulul ”Shipment”, in cadrul caruia produsul este transportat si predat clientului
- Modulul ”Customer Service”, care asigneaza un agent unei reclamatii primite din partea unui client, in urma unei livrari
- Modulul ”Marketing”, care serveste gasirea noilor clienti, prin organizarea campaniilor de marketing.

1.1 Modulul ”HQ”



Modulul ”HQ” are ca scop coordonarea departamentelor din cadrul companiei. In primul rand, din starea *init* se realizeaza tranzitia *initialize*, activand componentele ”Shop” si ”Localization”.

In urma realizarii unei vanzari in cadrul modulului ”Shop”, se plaseaza profitul in starea *profit*, care apoi se poate adauga la bugetul companiei, situat in starea *budget*, prin tranzitia *add profit to budget*. Pentru fiecare tranzactie, componenta ”Shop” adauga o entitate de tip *Shipment*, in starea *pending shipments*. Prin tranzitia *get buyer location* se plaseaza locatia cumparatorului in *buyer location*,

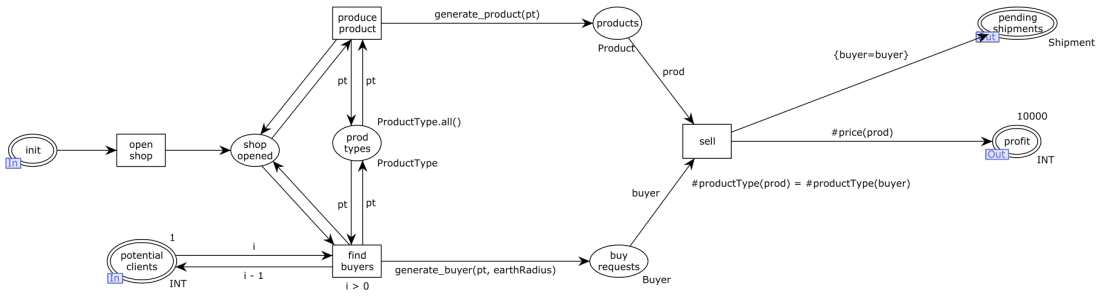
fiind input pentru modulul ”Localization”. Locatia reala a clientului nu ar fi cunoscuta inainte de localizarea GPS, insa este reprezentata in cadrul retelei, in scop demonstrativ.

In urma localizarii clientului, in starea *predicted location* se vor gasi coordonatele aproximative ale cumparatorului. Entitatea de tip *Shipment*, situata in *pending shipments* este concatenata cu locatia prezisa a clientului, returnand datele despre transport, in forma lor finala, in starea *shipment data*, prin tranzitia *prepare shipment*.

Apoi, modulul de ”Shipment” realizeaza livrarea, plasand in *delivery payment* plata transportului, achitata de cumparatorul produsului. Transportul se realizeaza cu ajutorul dronelor, situate in starea *drones*, care pot fi cumparate prin tranzitia *acquire drone*, daca permite bugetul companiei.

In urma livrarii, clientul poate face o reclamatie, stocata cu tipul *ClientFeedback*, in starea *complaints*. Reclamatiile sunt tratate de componenta ”Customer Service”. Agentii de customer service pot fi platiti prin tranzitia *donate to CS budget*, cu o suma prestabilita. In urma tratarii neintelegerii cu clientul, acesta va primi o compensatie din partea companiei, platita prin intermediul tranzitiei *pay compensations*. Pentru a gasi clienti noi, firma poate incepe o campanie de marketing, in cadrul modulului ”Marketing”. Clientii noi vor fi plasati in starea *potential clients*.

1.2 Modulul ”Shop”

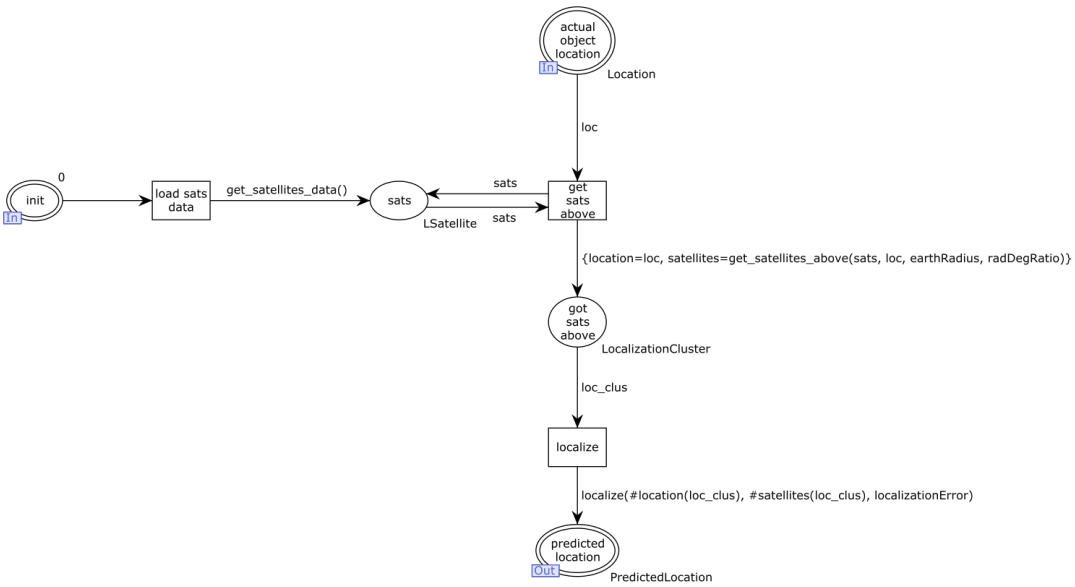


Modulul ”Shop” are ca scop generarea si comercializarea produselor oferite de

companie.

Mai intai, prin intermediul tranzitiei *open shop*, devine posibila generarea de noi produse si identificarea noilor clienti. Produsele sunt plasate in starea *products*, iar cumparatorii, in *buy requests*. Fiecarui produs i se atribuie un pret aleatoriu, generat pe baza unei distributii normale. Locatia clientilor este generata, de asemenea, aleatoriu. Latitudinea si longitudinea urmeaza distributii uniforme. Fiecare cumparator poate fi asociat cu un anumit tip de produs. In urma realizarii unei astfel de asocieri, prin tranzitia *sell*, se plaseaza noua livrare, in starea de output *pending shipments*. Profitul obtinut de companie se va afla in starea de output *profit*, urmand ca acesta sa fie adaugat la buget.

1.3 Modulul ”Localization”



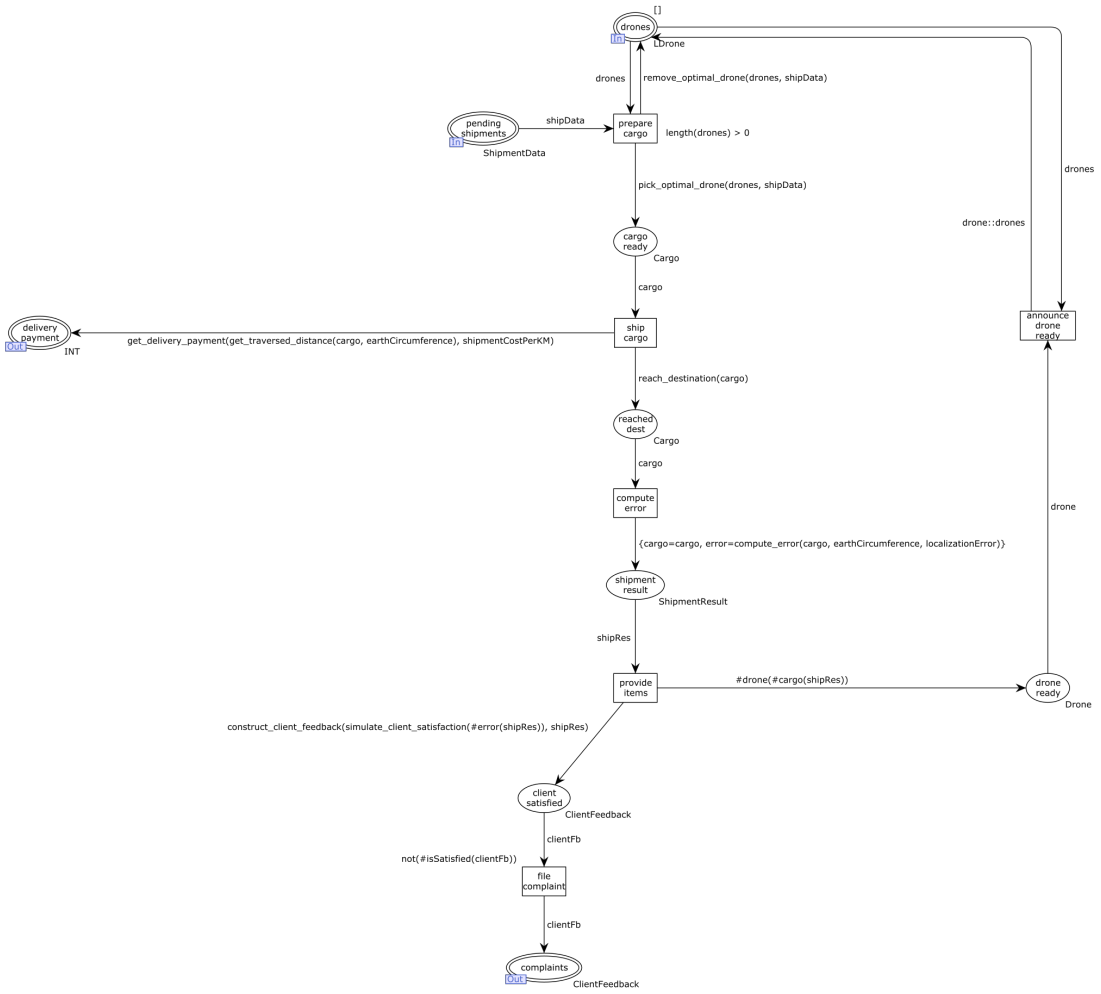
Scopul modulului ”Localization” este de a identifica locatia aproximativa a clientilor, pe baza satelitilor GPS.

Datele despre sateliti sunt returnate de functia *get_satellites_data*, generata automat in fisierul sursa ”satellites_data.sml”, creat cu ajutorul scriptului ”get_satellites_data.py”. Sunt stocate informatii reale despre pozitia satelitilor de GPS care orbiteaza in

jurul Pamantului (latitudine, longitudine, altitudine).

Locatiile satelitilor sunt plasate in starea *sats*, prin intermediul tranzitiei *load sats data*. Considerand locatia clientului (prezenta in *actual object location*, in scop demonstrativ), se sustrag satelitii aflati deasupra sa, doar acestia fiind capabili sa il localizeze. In urma executiei tranzitiei *localize*, satelitii de GPS aproximeaza locatia cumparatorului, plasand-o in starea de output *predicted location*. Aproximarea este supusa unei eroari generate aleatoriu uniform, per fiecare satelit.

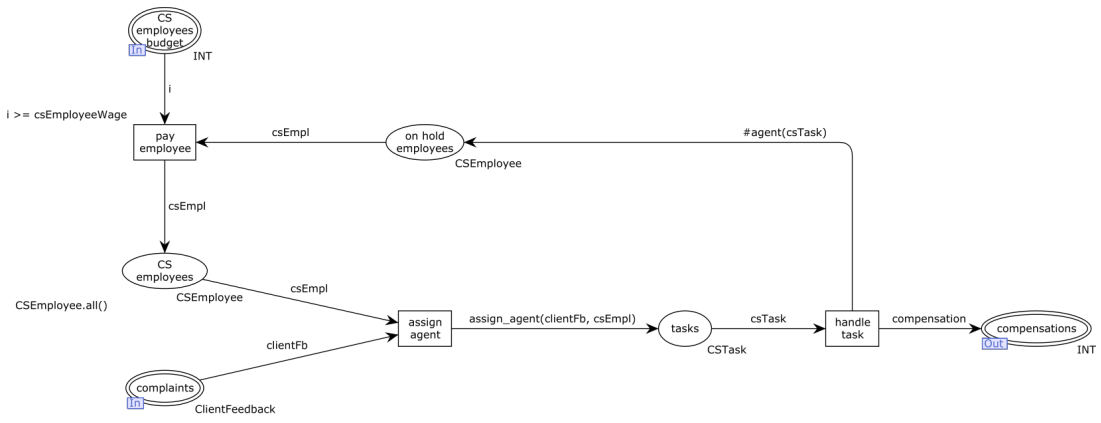
1.4 Modulul ”Shipment”



Modulul ”Shipment” trateaza livrarea produselor catre client, cu ajutorul dronelor.

Locatiile de input pentru aceasta componenta sunt *pending shipments* si *drones*, continand date referitoare la transport si o lista cu dronele disponibile. In urma tranzitiei *prepare cargo*, se asigneaza drona cea mai apropiata de client, generandu-se o entitate de tip *Cargo*. In cadrul tranzitiei *ship cargo* se calculeaza pozitia finala a dronei. Pozitia finala depinde de eroarea rezultata din aproximarea pozitiei cumparatorului, in modulul ”Localization”. De asemenea, se calculeaza costul transportului, suma platita de client fiind stocata in locatia de output *delivery payment*, urmand ca aceasta sa fie adaugata la bugetul companiei. Eroarea este calculata in cadrul tranzitiei *compute error*, plasand rezultatul transportului in *shipment result*. In urma executiei actiunii *provide items*, drona este stocata in *drone ready*, urmand ca aceasta sa fie readaugata in locatia *drones*, prin tranzitia *announce drone ready*. Feedback-ul clientului este stocat in starea *client satisfied*. Acesta depinde de eroarea in aproximarea locatiei in care este livrat produsul. In cazul in care clientul este nemultumit, acesta poate trimite o reclamatie companiei, prin *file complaint*. Reclamatiile vor fi tratate de echipa ”Customer Service”.

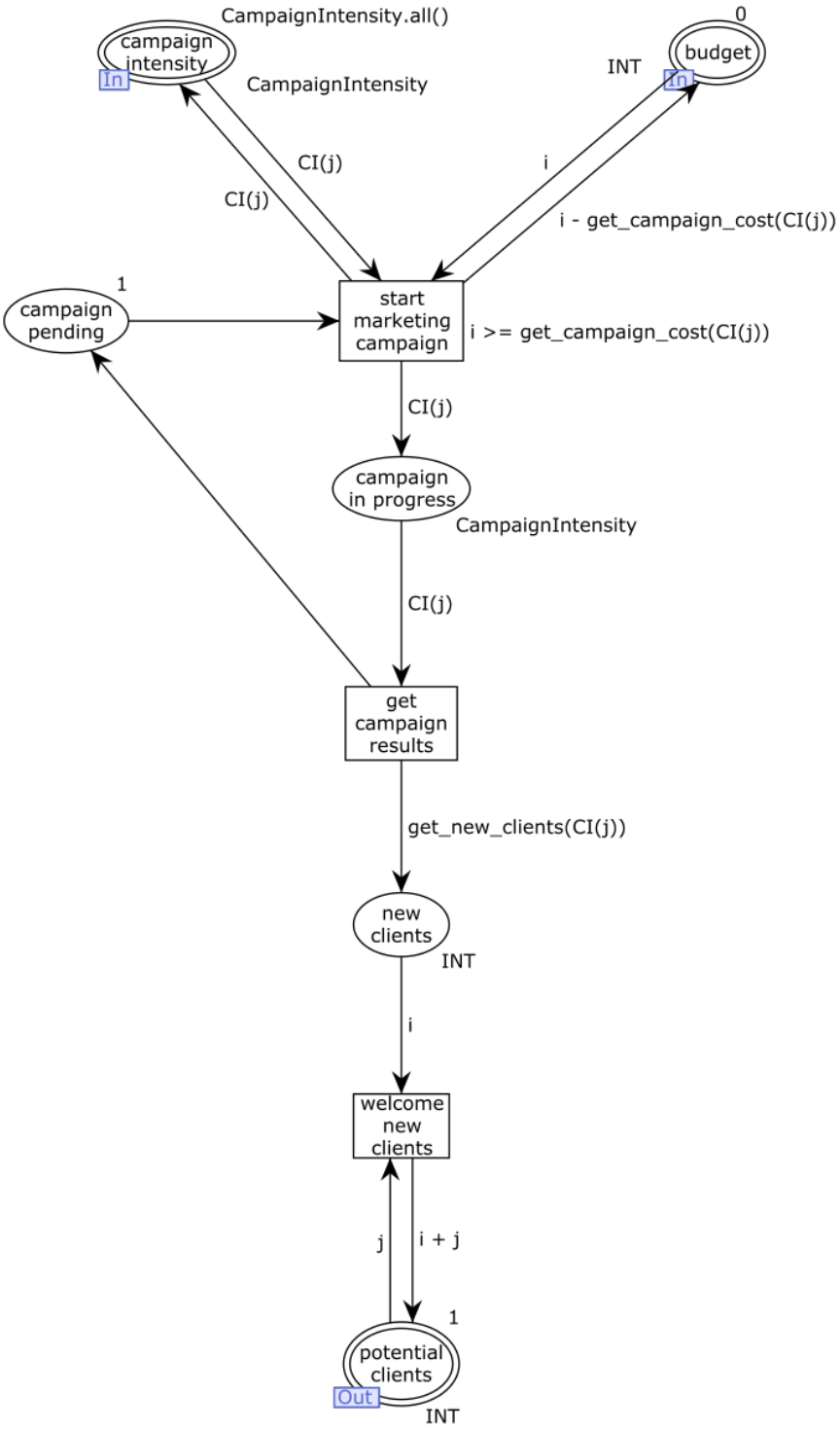
1.5 Modulul ”Customer Service”



Modulul ”Customer Service” trateaza reclamatiile primite de la clienti in urma livrarilor.

Reclamatiile vor fi stocate in locatia *complaints*. Daca sunt agenti disponibili in *CS employees*, se asigneaza unul dintre ei respectivei reclamatii, prin tranzitia *assign agent*, generandu-se in *tasks* o entitate de tip *CSTask*. In urma tratarii unui task, clientului i se promite o compensatie, care va fi achitata, pe baza bugetului companiei, in modulul ”HQ”. Dupa finalizarea unui task, angajatul este plasat in locatia *on hold employees*. Bugetul asignat agentilor de customer service se afla in starea de input *CS employees budget*. Angajatul nu va fi disponibil pentru a trata o alta reclamatie pana cand nu este platit de catre companie, prin tranzitia *pay employee*.

1.6 Modulul ”Marketing”



Scopul modulului de ”Marketing” este atragerea noilor clienti, in urma campaniilor de marketing.

O singura campanie poate fi activa la un moment dat. Locatia *campaign pending* asigura acest fapt. In urma executiei tranzitiei *start marketing campaign*, o noua campanie este initializata. Intensitatea unei campanii determina numarul de clienti care vor fi convinsi spre a cumpara produsele companiei. Campaniile mai eficiente vor implica cheltuieli mai mari din partea companiei. Prin executia tranzitiei *get campaign results*, se plaseaza numarul de clienti noi in locatia *new clients*, acest numar fiind calculat pe baza unei distributii uniforme. Actiunea *welcome new clients* adauga noii client in *potential clients*, aceasta fiind locatie input pentru modulul ”Shop”.

Chapter 2

Declaratii

2.1 Colset-uri

Colset-urile declarate in cadrul modelului sunt urmatoarele:

- Location = record latitude:REAL * longitude:REAL * altitude:REAL
- PredictedLocation = record original:Location * predicted:Location
- Satellite = record id:INT * name:STRING * latitude:REAL * longitude:REAL * altitude:REAL
- LSatellite = list Satellite
- LocalizationCluster = record location:Location * satellites:LSatellite
- ProductType = index P with 1..10
- Product = record productType:ProductType * price:INT
- Buyer = record productType:ProductType * location:Location
- Shipment = record buyer:Buyer
- ShipmentData = record shipment:Shipment * predictedLocation:Location

- Drone = record id:INT * location:Location
- LDrone = list Drone
- Cargo = record drone:Drone * shipData:ShipmentData
- ShipmentResult = record cargo:Cargo * error:REAL
- ClientFeedback = record isSatisfied:BOOL * shipRes:ShipmentResult
- CSEmployee = index E with 1..10
- CSTask = record clientFb:ClientFeedback * agent:CSEmployee
- CampaignIntensity = index CI with 1..10.

2.2 Variabile

Variabilele folosite in cadrul modelului sunt urmatoarele:

- loc_clus: LocalizationCluster
- sats: LSatellite
- loc: Location
- predLoc: PredictedLocation
- pt: ProductType
- prod: Product
- buyer: Buyer
- i, j, k: INT
- drone: Drone

- drones: LDrone
- shipment: Shipment
- shipData: ShipmentData
- cargo: Cargo
- shipRes: ShipmentResult
- clientFb: ClientFeedback
- csEmpl: CSEmployee
- csTask: CSTask

2.3 Constante

Constantele folosite in cadrul modelului sunt urmatoarele:

- radDegRatio = 0.0174532925
- earthRadius = 6371.0
- earthCircumference = 2.0 * Math.pi * earthRadius
- localizationError = 0.015 / earthCircumference
- minPrice = 50
- averagePrice = 1000
- droneCost = 1000
- shipmentCostPerKM = 0.25
- compensation = 500
- csEmployeeWage = 100

2.4 Functii

In continuare vor fi prezentate functiile folosite in cadrul modelului.

```
fun get_satellites_data() =
let
  val satellites_data = [
    {
      id=(24876),
      name="NAVSTAR␣43␣(USA␣132)",
      latitude=(~28.1175),
      longitude=(~64.4965),
      altitude=(20187.8447)
    },
    {
      id=(26407),
      name="NAVSTAR␣48␣(USA␣151)",
      latitude=(~53.6636),
      longitude=(~37.5563),
      altitude=(19786.5505)
    },
    {
      id=(27704),
      name="NAVSTAR␣52␣(USA␣168)",
      latitude=(~47.553),
      longitude=(55.5375),
      altitude=(19776.6716)
    }
  ]
in
  satellites_data
```

```
28 end;
```

Listing 2.1: Returneaza pozitiile satelitilor de GPS

```
1  fun is_satellite_above(sat: Satellite, loc: Location,
       earthRadius: real, degRadRatio: real) : bool =
2  let
3      val sat_xyz = lla_to_xyz(#latitude(sat), #longitude(sat),
           #altitude(sat), degRadRatio);
4      val loc_xyz = lla_to_xyz(#latitude(loc), #longitude(loc),
           #altitude(loc) + 10.0, degRadRatio);
5      val x1 = #1(sat_xyz);
6      val y1 = #2(sat_xyz);
7      val z1 = #3(sat_xyz);
8      val x2 = #1(loc_xyz);
9      val y2 = #2(loc_xyz);
10     val z2 = #3(loc_xyz);
11     val x3 = 0.0;
12     val y3 = 0.0;
13     val z3 = 0.0;
14     val quadratic_a = sqr(x2 - x1) + sqr(y2 - y1) + sqr(z2 -
           z1);
15     val quadratic_b = 2.0 * ((x2 - x1) * (x1 - x3) + (y2 - y1)
            * (y1 - y3) + (z2 - z1) * (z1 - z3));
16     val quadratic_c = sqr(x3) + sqr(y3) + sqr(z3) + sqr(x1) +
           sqr(y1) + sqr(z1) + 2.0 * (x3 * x1 + y3 * y1 + z3 * z1)
            - sqr(earthRadius);
17     val delta = sqr(quadratic_b) - 4.0 * quadratic_a *
           quadratic_c;
18 in
19     delta < 0.0
```

```
20 end;
```

Listing 2.2: Verifica daca un satelit este deasupra unui reper

```
1  fun get_satellites_above([]: LSatellite, loc: Location,
       earthRadius: real, degRadRatio: real) : LSatellite = []
2      | get_satellites_above(sat::sats_t: LSatellite, loc:
           Location, earthRadius: real, degRadRatio: real) :
           LSatellite =
3  let
4      val satsAbove = get_satellites_above(sats_t, loc,
           earthRadius, degRadRatio);
5  in
6      if is_satellite_above(sat, loc, earthRadius, degRadRatio)
           then
7          sat::satsAbove
8      else
9          satsAbove
10 end;
```

Listing 2.3: Returneaza o lista cu satelitii aflati deasupra unui reper

```
1  fun localize(loc: Location, []: LSatellite, err: real) :
       PredictedLocation =
2  let
3      val res_pred = {
4          latitude=0.0,
5          longitude=0.0,
```

```
6           altitude=0.0
7       };
8       val res_tuple = {
9           original=loc,
10          predicted=res_pred
11      };
12  in
13      res_tuple
14  end
15      | localize(loc: Location, sat::sats_t: LSatellite, err:
            real) : PredictedLocation =
16  let
17      val prev_pred = #predicted(localize(loc, sats_t, err));
18      val pred = {
19          latitude=uniform(#latitude(loc) - err * 180.0, #
                latitude(loc) + err * 180.0),
20          longitude=uniform(#longitude(loc) - err * 180.0, #
                longitude(loc) + err * 180.0),
21          altitude=uniform(#altitude(loc) - err, #altitude(loc)
                + err)
22      };
23      val res_pred = {
24          latitude=(#latitude(prev_pred) * Real.fromInt(length(
                sats_t)) + #latitude(pred)) / (Real.fromInt(length(
                sats_t)) + 1.0),
25          longitude=(#longitude(prev_pred) * Real.fromInt(length
                (sats_t)) + #longitude(pred)) / (Real.fromInt(
                length(sats_t)) + 1.0),
26          altitude=(#altitude(prev_pred) * Real.fromInt(length(
                sats_t)) + #altitude(pred)) / (Real.fromInt(length(
                sats_t)) + 1.0)
27      };
28      val res_tuple = {
```

```
29          original=loc,
30          predicted=res_pred
31      };
32  in
33      res_tuple
34  end;
```

Listing 2.4: Aproximeaza locatia unui obiect, pe baza observatiilor mai multor sateliti GPS

```
1  fun generate_product(pt: ProductType) : Product =
2  let
3      val price = abs(round(normal(Real.fromInt(averagePrice -
          minPrice), 100000.0))) + minPrice;
4      val product = {
5          productType=pt,
6          price=price
7      };
8  in
9      product
10 end;
```

Listing 2.5: Genereaza un produs, avand un anumit tip, cu un pret aleatoriu, pe baza unei distributii normale

```
1  fun generate_buyer(productType: ProductType, earthRadius: real
       ) : Buyer =
2  let
```

```
3       val lat = uniform(~90.0, 90.0);
4       val long = uniform(0.0, 180.0);
5       val alt = earthRadius;
6       val loc = {
7           latitude=lat,
8           longitude=long,
9           altitude=alt
10      };
11      val buyer = {
12          productType=productType,
13          location=loc
14      };
15  in
16      buyer
17  end;
```

Listing 2.6: Genereaza un client care are nevoie de un produs cu un anumit tip, cu o locatie aleatorie, pe baza unor distributii uniforme

```
1   fun construct_shipment_data(shipment: Shipment, predLoc:
        Location) : ShipmentData =
2       {
3           shipment=shipment,
4           predictedLocation=predLoc
5       };
```

Listing 2.7: Construieste o entitate de tip ShipmentData, pe baza unui shipment si a unei locatii prezise de modulul de localizare

```
fun generate_drone(id: int, earthRadius: real) : Drone =
let
    val lat = uniform(~90.0, 90.0);
    val long = uniform(0.0, 180.0);
    val alt = earthRadius;
    val loc = {
        latitude=lat,
        longitude=long,
        altitude=alt
    };
    val drone = {
        id=id,
        location=loc
    };
in
    drone
end;
```

Listing 2.8: Genereaza o drona, cu o locatie aleatorie, pe baza unor distributii uniforme

```
fun construct_client_feedback(isSatisfied: bool, shipRes:
   ShipmentResult) : ClientFeedback =
let
    val clientFb = {
        isSatisfied=isSatisfied,
```

```
5           shipRes=shipRes
6       };
7   in
8       clientFb
9   end;
```

Listing 2.9: Construieste o entitate de tip ClientFeedback

```
1   fun simulate_client_satisfaction(error: real) : bool =
2   let
3       val randNum = uniform(0.0, 1.0);
4   in
5       randNum > error
6   end;
```

Listing 2.10: Genereaza satisfactia clientului, pe baza unei distributii uniforme, considerand eroarea de localizare

```
1   fun get_traversed_distance(cargo: Cargo, earthCircumference:
        real) : real =
2   let
3       val init_loc = #location(#drone(cargo));
4       val ship_loc = #location(#buyer(#shipment(#shipData(cargo)
            )));
5       val delta_lat = abs(#latitude(init_loc) - #latitude(
            ship_loc));
6       val delta_long = abs(#longitude(init_loc) - #longitude(
            ship_loc));
```

```
7     val delta_alt = abs(#altitude(init_loc) - #altitude(
          ship_loc));
8     val delta_lat_m = delta_lat * earthCircumference / 180.0;
9     val delta_long_m = delta_long * earthCircumference /
          180.0;
10    val dist = Math.sqrt(sqr(delta_lat_m) + sqr(delta_long_m)
          + sqr(delta_alt));
11 in
12    dist
13 end;
```

Listing 2.11: Calculeaza distanta parcursa de o drona in cadrul unui transport

```
1  fun get_delivery_payment(travDist: real, shipCostPerKM: real)
       : int =
2  let
3     val cost = floor(travDist * shipCostPerKM);
4  in
5     cost
6  end;
```

Listing 2.12: Calculeaza costul unui transport, in functie de distanta parcursa de drona

```
1  fun compute_error(cargo: Cargo, earthCircumference: real,
       localizationError: real) : real =
2  let
3     val drone_loc = #location(#drone(cargo));
```

```
4       val ship_loc = #location(#buyer(#shipment(#shipData(cargo)
            )));
5       val delta_lat = abs(#latitude(drone_loc) - #latitude(
            ship_loc));
6       val delta_long = abs(#longitude(drone_loc) - #longitude(
            ship_loc));
7       val delta_alt = abs(#altitude(drone_loc) - #altitude(
            ship_loc));
8       val delta_lat_m = delta_lat * earthCircumference / 180.0;
9       val delta_long_m = delta_long * earthCircumference /
            180.0;
10      val dist = Math.sqrt(sqr(delta_lat_m) + sqr(delta_long_m)
            + sqr(delta_alt));
11      val error = dist / localizationError;
12  in
13      error
14  end;
```

Listing 2.13: Calculeaza eroarea procentuala dintre locatia finala a dronei si locatia reala a clientului

```
1   fun reach_destination(cargo: Cargo) : Cargo =
2   let
3       val drone_loc = #location(#drone(cargo));
4       val pred_loc = #predictedLocation(#shipData(cargo));
5       val ship_loc = #location(#buyer(#shipment(#shipData(cargo)
            )));
6       val delta_lat = #latitude(drone_loc) - #latitude(pred_loc)
            ;
7       val delta_long = #longitude(drone_loc) - #longitude(
```

```
        pred_loc);
8     val delta_alt = #altitude(drone_loc) - #altitude(pred_loc)
        ;
9     val dest_loc = {
10        latitude=(#latitude(ship_loc) + delta_lat),
11        longitude=(#longitude(ship_loc) + delta_long),
12        altitude=(#altitude(ship_loc) + delta_alt)
13    };
14    val dest_cargo = {
15        drone={
16            id=(#id(#drone(cargo))),
17            location=dest_loc
18        },
19        shipData=(#shipData(cargo))
20    };
21 in
22    dest_cargo
23 end;
```

Listing 2.14: Calculeaza pozitia finala a dronei, dupa realizarea transportului, considerand eroarea de estimare a locatiei clientului

```
1  fun get_optimal_drone_index(drones: LDrone, shipData:
     ShipmentData) : int =
2  let
3      fun dist_list_map(drone: Drone) : real =
4      let
5          val dist = sqr(#latitude(#location(drone)) - #latitude
               (#predictedLocation(shipData))) +
6                          sqr(#longitude(#location(drone)) - #
```

```
                                longitude(#predictedLocation(
                                shipData))) +
7                           sqr(#altitude(#location(drone)) - #
                                longitude(#predictedLocation(
                                shipData)));
8       in
9           dist
10      end;
11
12      fun min_index([]: real list, index: int, minIndex: int,
           minVal: real) : int = minIndex
13          | min_index(distList_h::distList_t: real list, index:
              int, minIndex: int, minVal: real) : int =
14          if distList_h < minVal then
15              min_index(distList_t, index + 1, index, distList_h
                    )
16          else
17              min_index(distList_t, index + 1, minIndex, minVal)
                    ;
18
19      val distList = map dist_list_map drones;
20      val minDistIndex = min_index(tl(distList), 1, 0, hd(
           distList));
21  in
22      minDistIndex
23  end;
```

Listing 2.15: Returneaza indexul dronei care este cea mai apropiata de un anumit client

```
1  fun pick_optimal_drone(drones: LDrone, shipData: ShipmentData)
       : Cargo =
2  let
3      val optimalDroneIndex = get_optimal_drone_index(drones,
           shipData);
4      val optimalDrone = List.nth(drones, optimalDroneIndex);
5      val cargo = {
6          drone=optimalDrone,
7          shipData=shipData
8      };
9  in
10     cargo
11 end;
```

Listing 2.16: Creaza un obiect de tip Cargo, asignand drona cea mai apropiata de un anumit client

```
1  fun remove_optimal_drone(drones: LDrone, shipData:
       ShipmentData) : LDrone =
2  let
3      val optimalDroneIndex = get_optimal_drone_index(drones,
           shipData);
4      val optimalDrone = List.nth(drones, optimalDroneIndex);
5      val dronesRem = rm optimalDrone drones;
6  in
7      dronesRem
8  end;
```

Listing 2.17: Scoate dintr-o lista de drone, drona cea mai apropiata de un anumit client

```
1 fun assign_agent(clientFb: ClientFeedback, agent: CSEmployee)
     : CSTask =
2 let
3     val task = {
4         clientFb=clientFb,
5         agent=agent
6     };
7 in
8     task
9 end;
```

Listing 2.18: Genereaza o entitate de tip CSTask, asignand un agent unei reclamatii

```
1 fun get_campaign_cost(CI(i): CampaignIntensity) : int =
2     pow(2, i);
```

Listing 2.19: Calculeaza costul unei campanii de marketing, in functie de intensitatea ei

```
1 fun get_new_clients(CI(i): CampaignIntensity) : int =
2 let
3     val newClients = discrete(1, pow(2, i));
4 in
```

```
5       newClients
6   end;
```

Listing 2.20: Genereaza un numar aleatoriu de noi clienti, pe baza unei distributii uniforme, in functie de intensitatea campaniei

```
1   fun lla_to_xyz(lat: real, long: real, alt: real, degRadRatio)
        : (real * real * real) =
2   let
3       val lat_rad = lat * degRadRatio;
4       val long_rad = long * degRadRatio;
5       val x = alt * Math.cos(lat_rad) * Math.cos(long_rad);
6       val y = alt * Math.cos(lat_rad) * Math.sin(long_rad);
7       val z = alt * Math.sin(lat_rad)
8   in
9       (x, y, z)
10  end;
```

Listing 2.21: Transforma un tuplu de coordonate LLA in coordonate XYZ

```
1   fun sqr(x: real) : real=
2       x * x;
```

Listing 2.22: Returneaza patratul unui numar

```
1 fun pow(x: int, 0: int) : int = 1
2     | pow(x: int, y: int) : int =
3     x * pow(x, y - 1);
```

Listing 2.23: Ridica un numar la o anumita putere