**4. Implementarea aplicației**

În acest capitol vor fi oferite detalii concrete privind implementarea aplicației, pe baza modelării funcționale cu ajutorul modulelor descrise în capitolul 3, ce descrie arhitectura aplicației. În prima parte a capitolului este detaliată diagrama de stări și modul de implementare, urmate de detaliile privind implementarea modulului de recunoaștere a poziției, cu cele două etape: de invățare și de testare, și detalierea implementării modulului de detecție a mișcării.

**4.1. Modulul de recunoaștere a poziției**

**4.1.1. Diagrama de stari. Pachete si clase utilizate**

Figura următoare (Fig. 9) reprezintă diagrama de stări a modulului de recunoaștere a pozițiilor. Pentru fiecare stare posibilă, se evidențiază pachetele de clase implementate.

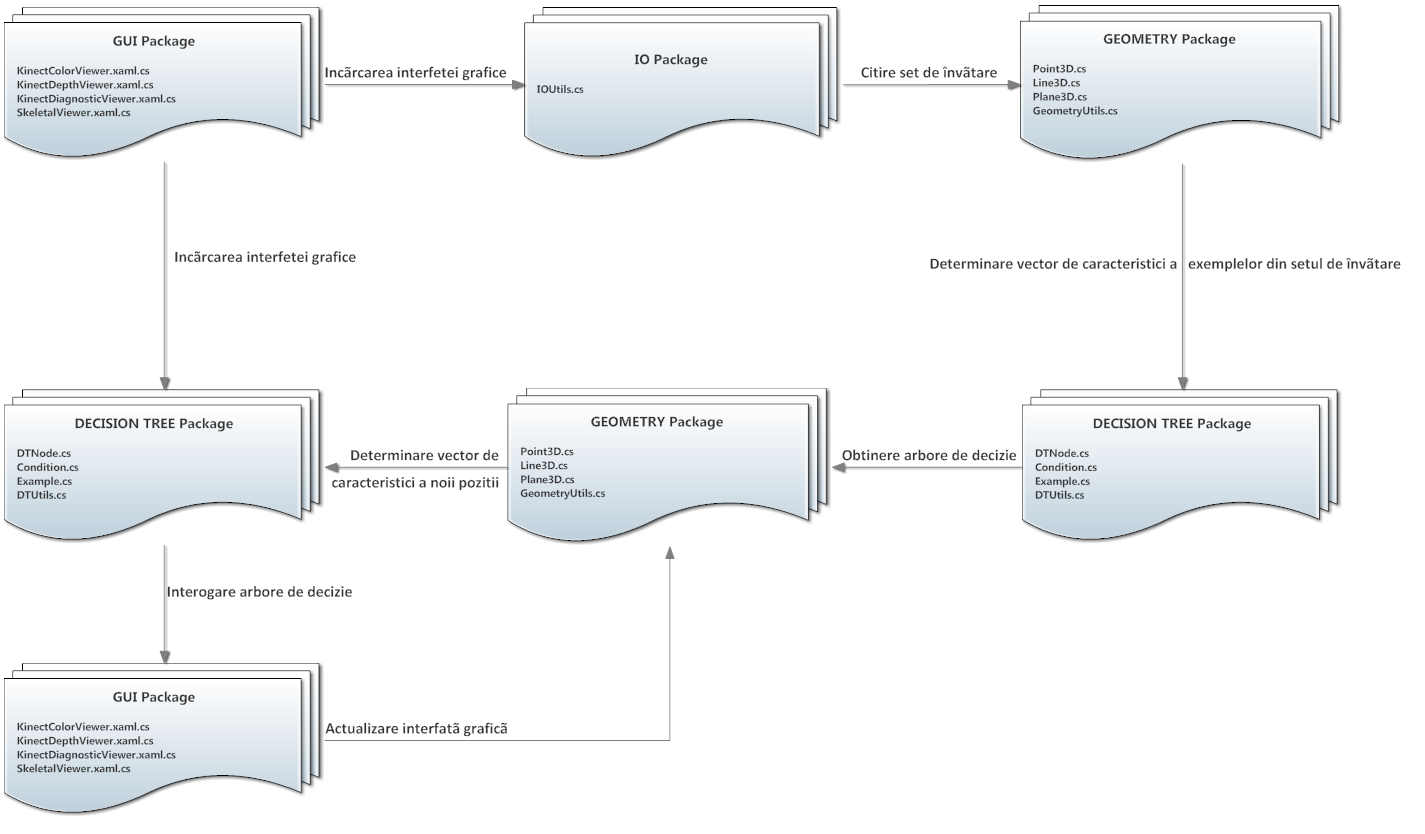


Fig. 9 – Diagrama de stări a modulului de recunoaștere a pozitiei. Pachetele și clasele implementate

TODO – modifica incarcarea interfetei grafice – dupa etapa de antrenare

După încărcarea interfeței grafice, aplicația primește ca date de intrare coordonate a 20 de articulații ale corpului uman de la senzorul Kinect, determinând un vector de caracteristici pentru fiecare componentă a corpului în parte. Ulterior, se creează un fir de execuție pentru fiecare arbore în parte și se realizează antrenarea acestuia. Dupa acest pas, aplicația parcurge o serie de pași, ce cuprind citirea unor noi coordonate de la senzorul Kinect, determinarea vectorilor de caracteristici ale acestora și interogarea fiecărui arbore de decizie în cadrul unor fire de execuție separate, furnizând în final poziția detectată. În funcție de poziția detectată și de cea precedentă, se actualizează sau nu etichetele corespunzătoare pozițiilor și mișcărilor componentelor corpului uman din interfața grafică.

. **Încărcarea interfeței grafice**

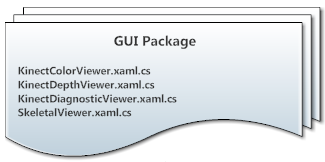


Fig. 10 – Pachete și clase implementate în încărcarea interfeței grafice

Primul pas al aplicației constă în încărcarea elementelor interfeței grafice. Interfața grafică realizată pentru afișarea poziției și mișcării detectate folosește o aplicație deja existentă, un exemplu de utilizare a funcțiilor din trusa de dezvoltare a dispozitivului Kinect (Kinect SDK), furnizată de Microsoft [11]. Aplicația furnizată odată cu programul de instalare a dispozitivului Kinect furnizează 3 imagini:

❖ *Color Image* - imaginea color a înregistrării video furnizată de dispozitiv, fără nicio detecție a corpului uman

❖ *Depth Image* - imaginea ce evidențiază corpul uman detectat, a cărui adâncime este marcată prin colorarea cu diferite nuanțe

❖ *Detected Skeletons* – imaginea minimalistă ce prezintă cele 20 de articulații, formând astfel scheletul corpului individului supravegheat

Alături de aceste 3 imagini, se adaugă etichete ale poziției și mișcării curent detectate, corespunzătoare fiecărei părți componente ale corpului.

**Citire set de antrenare**

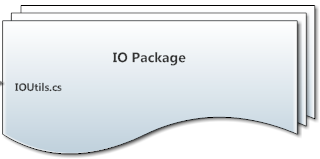


Fig. 11 – Pachete și clase implementate în citirea setului de antrenare

Antrenarea inițială a arborelui de decizie este realizată prin intermediul citirii dintr-un fișier de pe disc coordonatele specifice fiecărei poziții ce se dorește a fi recunoscută.

**Determinare vector de caracteristici a exemplelor din setul de antrenare**

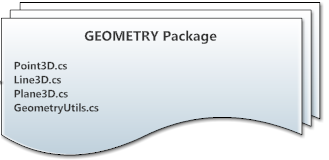


Fig. 12 – Pachete și clase implementate în citirea setului de antrenare

Datele preluate de la dispozitivul Kinect constau în 20 de coordonate 3D. Clasa ce modelează o intrare din setul de date este **Example.cs**, ce reține poziția rezultată (inițial nulă) și un dicționar ce mapează numele unui atribut cu valoarea acestuia, formând vectorul de caracteristici.

class Example

{

private Dictionary<String, Double> attributes;

private String position;

}

**Obținere arbore de decizie**



Fig. 13 – Pachete și clase implementate in obținerea arborelui de decizie

După formarea listelor de exemple caracteristice fiecărei părți componente ale corpului, se setează în nodurile rădăcină ale arborilor setul de antrenare din care se face clasificarea, lista de atribute și clase caracteristice părții componente.

Arborele se construiește în mod recursiv. Modelarea unui nod din arbore este realizată prin intermediul clasei **DTNode.cs**. Fiecărui nod îi corespunde o valoare rezultat (nulă dacă este un nod intermediar și o denumirea unei poziții dacă este un nod frunză), un set de antrenare alcătuit dintr-o listă de exemple, clasele specifice arborelui componentei corpului, lista de atribute din care se mai poate face clasificarea și un dicționar ce modelează descendenții acestui nod, unde cheia reprezintă nodul descendent, iar valoarea este marcată prin condiția de evaluare a atributului cu câstigul informațional cel mai mare din nod.

class DTNode

{

private String position;

private List<Example> trainingSet;

private List<String> attrNames;

private Dictionary<DTNode, Condition> children;

private List<String> classes;

}

Metodele de operare asupra arborelui de decizie sunt implementate în clasa **DTUtils.cs**. În cadrul acestei clase, sunt implementate metode de calcul al entropiei unui anumit atribut pentru o anumită partiționare a domeniului de valori, calcul al câștigului informațional unui atribut, determinare a unui nod trivial (implicit nod frunză), construcție și afișare a arborelui de decizie și determinarea clasei pentru un anumit exemplu de test.

class DTUtils

{

public static double getEntropy(List<Example>, List<String>, String, int);

public static GainResult getGain(List<Example>, String, List<String>);

public static bool isTrivial(DTNode);

public static void buildTree(DTNode);

public static void printTree(DTNode);

public static String findValueInTree(DTNode, Example);

}

**Determinarea vectorului de caracteristici a poziției curente**

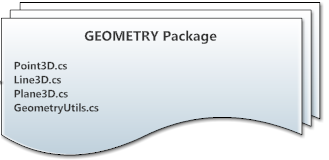


Fig. 14 – Pachete și clase implementate în determinarea vectorului de caracteristici al poziției curente

Determinarea vectorului de caracteristici al poziției curente se realizează în mod similar pasului 2 de citire a setului de date de antrenare.

**Interogarea arborelui de decizie**



Fig. 15 – Pachete și clase implementate în interogarea arborelui

Pentru determinarea poziției unui anumit vector de caracteristici calculat la pasul anterior, se apelează metoda **findValueInTree(DTNode, Example)** din cadrul clasei **DTUtils.cs**. Această metodă se apelează pentru fiecare arbore în parte în mod paralel, asemenea construcției arborilor. Poziția furnizată în final reprezintă reuniunea celor 5 poziții ale trunchiului și membrelor.

**Actualizarea interfeței grafice**

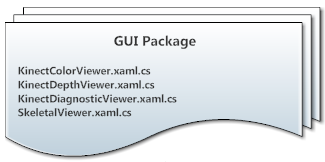


Fig. 16 – Pachete și clase implementate în actualizarea interfeței grafice

Interfața grafică se actualizează ori de câte ori una din poziții, și implicit miscări, se modifică. Atât cele trei ferestre de vizualizare a imaginilor, cât și etichetele pozițiilor și mișcărilor pot suferi modificări.

Recunoașterea poziției corpului uman necesită două etape:

❖ o etapă de învățare, în care se preiau coordonatele 3D furnizate de dispozitivul Kinect, se determină cu ajutorul acestora un vector de caracteristici și, ulterior, se obțin arborii de decizie pentru fiecare componentă a corpului în parte: trunchi, membre superioare, membre inferioare

❖ o etapă de testare, în care prima acțiune coincide cu etapa de învățare: preluarea datelor de test de la dispozitivul Kinect. Prin interogarea arborilor formați în etapa anterioară, se obțin pozițiile membrelor ce formează, prin reuniune, modelul complet al poziției corpului uman supravegheat.

**4.1.2. Etapa de învățare**

**Determinarea vectorului de caracteristici**

Primul pas din etapa de învățare constă în preluarea coordonatelor 3D de la dispozitivul Kinect. Pentru seria de poziții ce se doresc a fi învățate în final, sunt necesare doar 12 din cele 20 de poziții ale articulațiilor. Pentru rezultate bune pe orice individ, se evită folosirea distanțelor sau a pozițiilor fixe a articulațiilor drept caracteristici în etapa de învățare. Motiv pentru care, determinarea vectorului de caracteristici constă în calculul unghiurilor formate între dreapta formată de două puncte de articulație și unul dintre cele trei plane ortogonale: xOy, yOz, xOz, așa cum este ilustrat în Fig. 17.

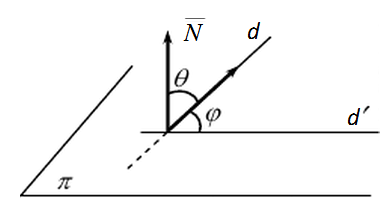


Fig. 17 – Unghiul determinat de o dreaptă cu un plan

Unghiul dintre o dreaptă și un plan este definit în geometria elementară ca fiind unghiul dintre dreaptă și proiecția ortogonală a acesteia pe plan, conform Fig. 15 și formulei (2) următoare:

(2)

O intrare din setul de învățare este modelată în clasa **Example.cs**. Această clasă reține poziția corespunzătoare și atributele caracteristice clasificării ulterioare, utilizate în construcția arborelui de decizie. Atributele acestuia sunt modelate cu un dicționar, unde cheia reprezintă numele atributului (unghiului), iar valoarea este valoarea acestuia.

**Obținerea arborelui de decizie**

Algoritmul de construcție a arborelui de decizie este un algoritm de construire inductivă. În final, după construcția arborelui, acesta trebuie să poată furniza la ieșire o valoare logică (decizie), având la intrare un exemplu descris prin valori ale unui set de atribute. Atributele, așa cum s-a menționat anterior, sunt elemente ale unui dicționar, format cu ajutorul vectorului de caracteristici determinat în pasul precedent. Fiecare nod din arbore reprezintă un test al valorii unui atribut reprezentat de un unghi, iar arcele din arbore corespund valorilor unghiului după partiționarea valorilor cu câștigul informational cel mai mare. Nodurile frunză specifică decizia la care se ajunge pentru calea care duce la acel nod, respectiv poziția componentei corpului corespunzătoare arborelui. Deoarece algoritmul ID3 este limitat în cazul în care atributele clasificatoare sunt de tip numeric și aparțin unui domeniu larg de valori, am folosit o extindere C4.5 a algoritmului, detaliată în capitolul 4.1.

La fiecare pas din extinderea arborelui de decizie, se selectează atributul ce diferentiază cel mai mult exemplele încă neclasificate. Cu alte cuvinte, se selectează atributul ce aduce câștigul informațional cel mai mare. Câștigul informațional este determinat în urma calculului entropiei fiecărui domeniu de valori pe care îl poate lua atributul respectiv. După selectarea acestui atribut, se divid exemplele din setul de învățare în două subseturi, corespunzătoare valorilor posibile ale atributului. Construcția arborelui este o construcție recursivă, apelându-se în fiecare nod din arbore pentru subseturile obținute și lista de atribute rămase. Procedeul recursiv se termină când toate exemplele dintr-un subset au aceeași clasificare, nodul devenind trivial în informație. Dacă se termină atributele și subsetul conține încă exemple cu clase diferite, atunci se selectează probabilistic poziția rezultantă, luând în considerare doar exemplele din setul de învățare rămas.

**4.1.3. Etapa de testare**

**Determinarea vectorului de caracteristici**

Procesul determinării valorilor atributelor în funcție de care va fi realizată clasificarea este similar determinării vectorului de caracteristici din etapa de învățare.

**Interogarea arborelui de decizie**

Etapa de testare folosește arborele construit în etapa de învățare și furnizează o poziție finală prin parcurgerea arborelui în adâncime, respectând condițiile impuse pe fiecare arc.

Fiind o aplicație ce furnizează o ieșire (poziția corpului uman) în funcție de datele preluate de la dispozitivul Kinect, resursa timp reprezintă un factor foarte important. Motiv pentru care atât construcția arborilor, cât și parcurgerea acestora se realizează în mod paralel, corespunzătoare câte unui fir de executie separat. Numărul de atribute maxim este 5, în cazul arborelui de decizie corespunzător trunchiului. In cazul membrelor inferioare și superioare numărul acestora este de 4 atribute. Înălțimea unui arbore de decizie depinde atât de numărul de atribute ce realizează partiționarea, cât și de numărul de refolosire a unui atribut in clasificare. Cu toate acestea, timpul de furnizare a poziției unei componente a corpului individului supravegheat este foarte mic.

**Obținerea poziției**

Pentru interogarea arborilor se creează câte un fir de execuție pentru fiecare arbore în parte. După așteptarea fiecăruia dintre aceste 5 fire de execuție, se compune poziția finală a corpului individului supravegheat, formată din reuniunea poziției trunchiului, membrului superior stâng, membrului superior drept, membrului inferior stâng, membrului inferior drept.

Prioritățile de execuție, atât a firelor ce creează cei 5 arbori, cât și a firelor ce îi interoghează pentru determinarea poziției componentei corpului, urmăresc un model de lucru cooperativ – ele partajează timpul, și nu resursele. Fiecare arbore este format și interogat folosind propriile resurse. Din acest motiv, sincronizarea acestor fire de execuție își propune doar un punct final comun.

**4.1.4. Modificări aduse algoritmului ID3**

Datorită faptului că arborele de decizie necesar în clasificare este construit pe baza unor atribute continue, algoritmul ID3 trebuie extins pentru a putea fi construit și pe baza unor valori numerice reale, cu optimizări aduse în acest caz.

Așa cum este menționat în articolul [12] realizat de Hunt si colegii săi, algoritmul C4.5 utilizează o abordare *„divide și cucereste”* (*„divide-et-conquer”*). Algoritmul generează un arbore de decizie pornind de la un set de date D.

❖ dacă setul de date D corespunzător unui nod din arbore îndeplinește condiția de oprire, atunci nodul curent devine frunză etichetată cu clasa cea mai frecventă în set. Condiția de oprire constă în atingerea unui prag *θ* de apariție a unei anumite clase în setul de antrenare.

Fie atributul *Ci* cu un domeniu continuu de valori. Pentru a realiza o partiție cât mai eficientă, se examinează valorile atributului în întregul set de antrenare corespunzător nodului curent din arbore și se ordonează crescător. Pentru fiecare caz în care două valori alăturate ale atributului, fie ele pe pozițiile *i* și *i+1*, aparțin unei poziții diferite, se partiționează temporar setul de învățare în două seturi: setul cu exemple ale căror valori ale atributului *Ci* sunt mai mici decât valoarea atributului de pe poziția *i* și setul exemplelor ale căror valori ale atributului *Ci* sunt mai mari sau egale cu valoarea atributului de pe poziția *i*. Pentru fiecare astfel de partiționare, se calculează câștigul informațional și se calculează acea partiție care maximizează acest castig. Cu cât zgomotul din exemplele setului de antrenare este mai redus, cu atât scade numărul de partiționări posibile pentru fiecare atribut. Astfel, se garantează că la fiecare extindere a arborelui, numărul de descendenți va fi mereu 2 (partiționare binară).

În forma generală a unui algoritm ID3, în nodul ce se dorește extins în pasul curent, se determină atributul ce aduce un câștig informațional cât mai mare. Ulterior, acesta se elimină din lista atributelor ce va realiza clasificarea în modurile descendente. Comparația dintre cele două abordări este ilustrată în Fig. 18 și Fig. 19, în care se dorește clasificarea poziției trunchiului. Acest caz a demonstrat atât teoretic, cât și practic, creșterea ratei de succes a rezultatelor furnizate în cazul în care atributele dintr-un nod curent nu se elimină, ci persistă în descendenți pentru un număr de *x* ori, unde *x* poate fi determinat empiric.

Se notează cu *„A.\*”* numele atributului generic ce realizează clasificarea.

Atributul *„A.I.J.K”* reprezintă unghiul determinat de pozițiile celor 3 articulații: I, J și K, unde I,J,K ϵ {0,1,2,…, 19}. Indicii articulațiilor sunt aceleași din Fig. 7, în capitolul “Arhitectura aplicației”.

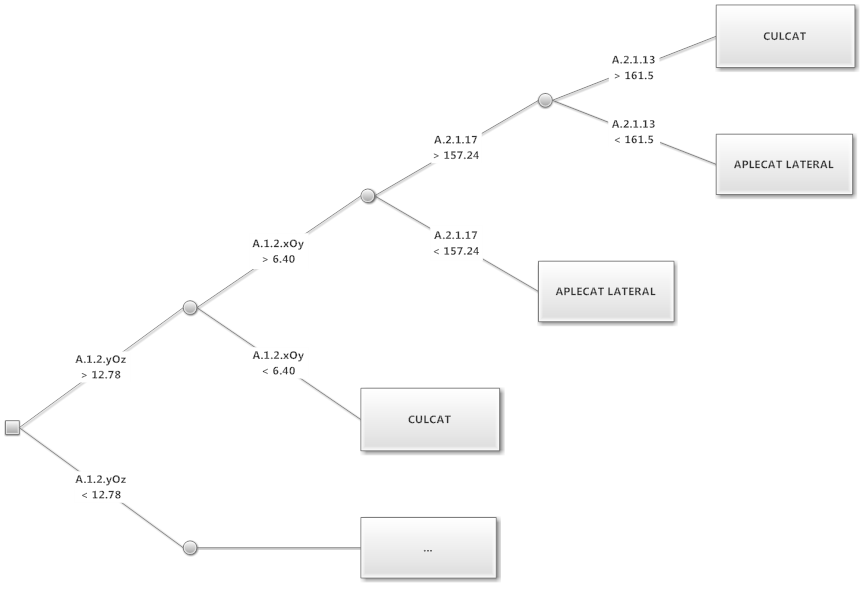


Fig. 18 – Arborele obținut folosind un algoritm ID3 fără extinderi

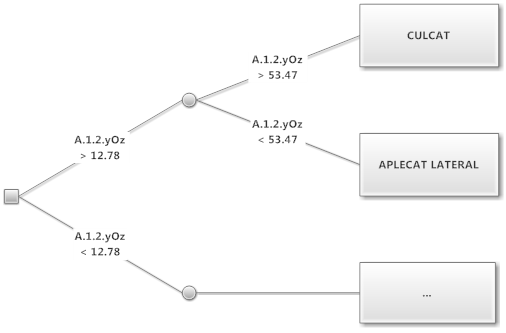


Fig. 19 – Arborele obținut folosind un algoritm ID3 cu extinderi C4.5

Nodul stâng al rădăcinii, corespunzător unor date de antrenare cu valori ale unghiului format între articulația 1 și 2 mai mari decât 12.78, conține în setul de antrenare doar exemple ale căror poziție este „culcat” sau „aplecat lateral”. Atributele ce au rămas în nodul stâng nu sunt cele mai elocvente. Atributul ce face cu adevărat diferențierea între cele două poziții este atributul anterior eliminat. Motiv pentru care, pentru rezultate cât mai aproape de cele corecte și o creștere a ratei de succes a aplicației, atributele persistă în extinderea arborelui un număr fix de ori. Rezultatele cele mai bune s-au obtinut în cazul posibilității de reutilizare în clasificare a unui atribut de 2 ori.

Ca urmare, subarborele clasifică setul de antrenare folosind în mod succesiv atributul *A.1.2.yOz* (unghiul format de dreapta articulațiilor 1 și 2 și planul elementar de coordonate ortogonale yOz). Astfel, subarborele stâng al rădăcinii are o adâncime mai mică cu două unități, fiind parcurs într-un număr mai mic de unități de timp.

**4.2 Modulul de detecție a miscării**

Modulul de detecție a mișcării folosește ca dată de intrare poziția recunoscută în cadrul modulului precedent. În momentul în care poziția curentă a cel puțin uneia din părțile componente ale corpului uman s-a modificat, se realizează o tranziție din starea curentă într-o nouă stare incidentă. Actualizarea acestei stări depinde de ultima poziție recunoscută. Dacă noua stare este finală, se obține rezultatul mișcării detectate. Odată cu efectuarea stranziției si modificarea stării curente, se actualizează și ponderile tuturor tranzitiilor din această stare (ponderea tranziției efectuate crește, în timp ce ponderea tranzițiilor disjuncte scad).

**4.2.1 Diagrama de stări. Pachete și clase implementate**

Figura următoare (Fig. 20) reprezintă diagrama de stări a modulului de detecție a mișcării corpului. Pentru fiecare stare posibilă, se evidențiază pachetele de clase implementate.

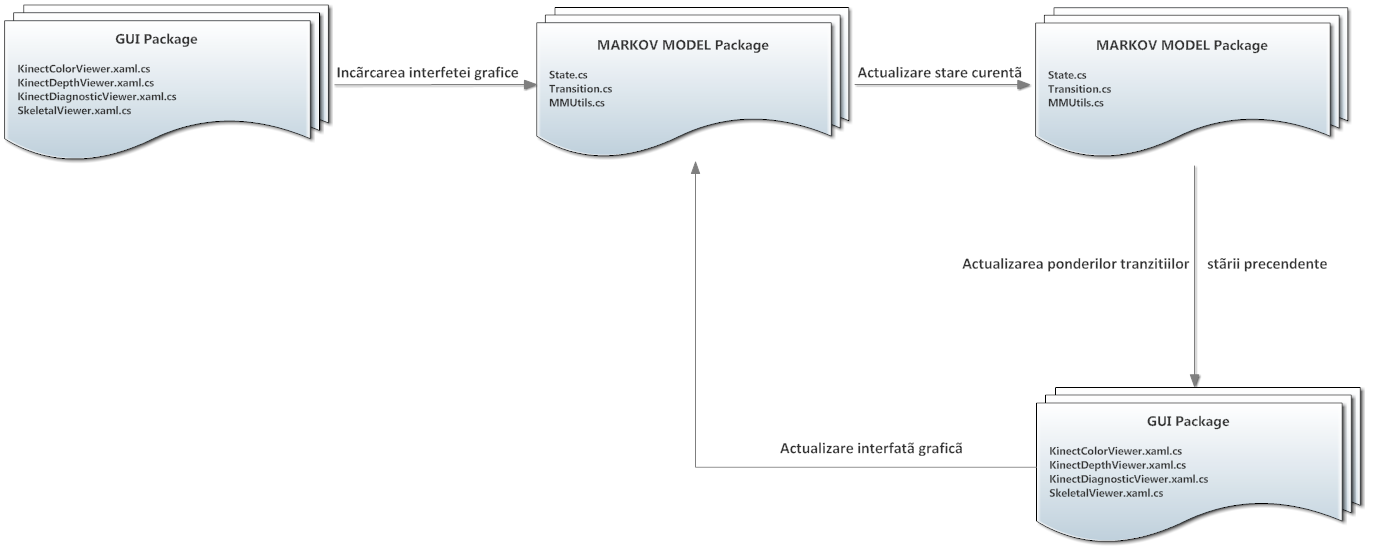
****

Fig. 20 – Diagrama de stări a modulului de detecție a mișcării. Pachetele și clasele implementate

Modulul de detecție a mișcării corpului uman este parcurs imediat după etapa de învățare a modulului de recunoaștere a poziției corpului, în același timp cu etapa de testare. Procesul de încărcare a interfeței grafice reprezintă punctul de start atât pentru modulul de detecție, cât și pentru etapa de testare din cadrul modulului de recunoaștere a poziției.

**Actualizarea stării curente**



Fig. 21 - Pachete și clase implementate în actualizarea stării curente

Actualizarea stării curente din cadrul modelului Markov folosește clasele implementate în pachetul **Markov Model**. O starea din cadrul modelului Markov este modelată în **State.cs**. Aceasta este caracterizată printr-un nume, un tip (stare inițială, stare intermediară și stare finală), un rezultat (dacă starea este finală, atunci rezultatul reține mișcarea produsăș in caz contrar, este un sir vid) și un dicționar de tranziții, mapate după identificatorii acestora.

class State

{

private String name;

Dictionary<String, Transition> transitions;

Constants.stateType type;

String motion;

}

Tranziția este modelată de clasa **Transition.cs**. O tranziție deține un identificator (după care se realizează maparea din dictionarul de tranziții dintr-o stare), o probabilitate de apariție și starea următoare corespunzătoare tranziției respective.

class Transition

{

private double count;

private double totalCount;

private String action;

private State newState;

}

Instanțele claselor anterior menționate (stări și tranziții) sunt folosite în cadrul clasei **MMUtils.cs**, în care se inițializează modelele Markov pentru fiecare parte componentă a corpului și se obține următoarea stare a modelului în urma unui eveniment precizat ca parametru metodei.

class MMUtils

{

Dictionary<String, State> bodyStates;

Dictionary<String, State> handStates;

Dictionary<String, State> legStates;

public State getNextBodyState(String stateName, String action);

public State getNextHandState(String stateName, String action);

public State getNextLegState(String stateName, String action);

public Dictionary<String, State> initBodyStates();

public Dictionary<String, State> initHandStates();

public Dictionary<String, State> initLegStates();

}