

 **Facultatea de Automatică și Calculatoare**

**Programul de Licență: Ingineria Sistemelor**

**Aplicație software de recunoaștere a stărilor emoționale bazate pe trasaturile faciale**

**Proiect de diplomă**

Florentina-Mihaela BLIDARIU

Conducători științifici:

Ș.l.dr.ing. Loredana STANCIU (Universitatea Politehnica Timişoara)

**Timișoara**

**2016**

**Cuprins**

[**1. Introducere** 5](#_Toc453012058)

[***1.1Definirea problemei și motivarea alegerii temei*** 5](#_Toc453012059)

[***1.2Scopul final al proiectului*** 6](#_Toc453012060)

[***1.3Prezentarea pe scurt a capitolelor existente în lucrare*** 6](#_Toc453012061)

[**2. Fundamentare teoretică** 8](#_Toc453012062)

[***2.1 Scurtă prezentare a mecanismelor folosite în implementare*** 8](#_Toc453012063)

[***2.2 Biblioteca OpenCV*** 8](#_Toc453012064)

[Recunoașterea **[ edit ]** 12](#_Toc453012065)

[**Metode sistem de vizualizare de calculator [ edit ]** 14](#_Toc453012066)

[***2.3 Haarcascade training*** 17](#_Toc453012067)

[***2.4 Limbajul Python*** 17](#_Toc453012068)

[***2.4.1 Structuri de control în Python*** 18](#_Toc453012069)

[***2.5 Limbajul XML*** 20](#_Toc453012070)

[***2.5.1 Scheme XML prezente în lucrare*** 22](#_Toc453012071)

[**3. Specificațiile aplicației** 23](#_Toc453012072)

[***3.1 Schema bloc a sistemului. Scurtă descriere a aplicației*** 23](#_Toc453012073)

[***3.2 Funcțiile sistemului*** 28](#_Toc453012074)

[***3.3 Interfața cu utilizatorul*** 30](#_Toc453012075)

[***3.4 Structuri de date și fișiere*** 31](#_Toc453012076)

[***3.4.1 Fișiere*** 31](#_Toc453012077)

[***3.4.2 Structuri de date*** 33](#_Toc453012078)

[**4. Proiectarea de detaliu** 35](#_Toc453012079)

[***4.1 Arhitectura programului*** 35](#_Toc453012080)

[***4.2 Descrierea componentelor*** 35](#_Toc453012081)

[***4.2.1 Fișierele PXA*** 37](#_Toc453012082)

[***4.2.2 Dicționare Pxa*** 39](#_Toc453012083)

[***4.2.3 Reguli pentru crearea dicționarelor*** 41](#_Toc453012084)

[***4.2.4 FișiereXLS*** 44](#_Toc453012085)

[***4.2.5 Dicționare Qml*** 46](#_Toc453012086)

[***4.2.6 Qml String*** 47](#_Toc453012087)

[***4.2.7 USERCOMAND*** 48](#_Toc453012088)

[**5. Utilizarea sistemului** 51](#_Toc453012089)

[***5.1 Scurtă prezentare*** 51](#_Toc453012090)

[***5.2 Instalare*** 51](#_Toc453012091)

[***5.3 Utilizare*** 52](#_Toc453012092)

[***6. Realizarea, punerea în funcțiune și rezultate experimentale*** 57](#_Toc453012093)

[**7. Concluzii** 60](#_Toc453012094)

[**8.Bibliografie** 61](#_Toc453012095)

# **1. Introducere**

## ***1.1Definirea problemei și motivarea alegerii temei***

Prezenta lucrare reprezintă o aplicație software de recunoaștere a stărilor emoționale pe baza trasăturilor faciale, fiind destinată utilizatorilor windows dar și utilizatorilor android. Această aplicație are capacitatea de a recunoște stările emoționale în urma unor capturi efectuate de către utilizator(camera web sau camera telefonului personal). În funcție de trăsaturile faciale ale persoanei din poză, aplicația poate descrie starea emoțională a resprectivei persoane.

În ultima perioadă, s-a căutat găsirea unor soluții prin care omul să interacționeze cu calculatoarele, iar calculatoarele la rândul lor să interacționeze cu oamenii. Domeniul în cadrul căruia aceste lucruri sunt posibile se numește *affective computing*. Acest domeniu presupune recunoașterea, interpretarea, procesarea și simularea emoțiilor umane. De la descoperirea acestui domeniu, s-au realizat mai multe aplicații prin care sistemul sa poată interacționa cu omul. Ducerea la bun sfârșit a acestor aplicații a fost posibilă datorită omului, dar folosindu-se de unele dintre cele mai importante științe cum ar fi: *Computer* *Science*, *Psychology* și *Cognitive Science*.

Aplicațiile realizate pot răspunde emoțiilor omenești, dar și altor stimuli externi. Un dispozitiv cu asemenea capacități, folosește o varietate de surse pentru determinarea emoțiilor. Gesturile, expresiile faciale, vocea sunt doar câteva dintre schimbările semnificative care pot afecta starea emoțională a unei persoane, acestea pot fi detectate și interpretare de către un calculator.

Se poate spune că recunoaşterea facială este o tehnică biometrică fiind folosită pentru identificarea unei persoane dar și pentru identificarea stărilor emoționale ale acelei persoane. Sistemul de recunoaştere facială se bazează pe imaginea statică a feţei unei persone(o fotografie) care este de fapt un set de pixeli ordonaţi după un anumit model (pixelul fiind unitatea funcţională fundamentală a unei imagini digitale).

Sistemul de recunoaştere facială percepe chipul unui individ ca pe o mulţime de pixeli alăturaţi. Abilitatea sistemului de localizare a feţei individului şi nu a imaginilor de fond este esenţială în procesul de recunoaştere facială.

Înainte de procesul propriu-zis de recunoaştere este necesară crearea unei galerii de imagini, galeria fiind un set de modele biometrice care serveşte drept referinţă în procesul de comparare. Crearea galeriei de imagini presupune parcurgerea următoarelor etape: captarea imaginii, detectarea feţei, standardizarea, extragerea trăsăturilor şi crearea şablonului pentru fiecare imagine.

## ***1.2Scopul final al proiectului***

Acest subiect nu este unul foarte abordat datorita faptului că nu este întocmai ușor de realizat până acum și nu prea există variante cu care ar putea fi comparat. Are o mare importanță pentru că realizarea acestei aplicații constituie cunoașterea stărilor emoționale ale unui individ doar din poze, fără a fi nevoie de o altă analiză.

Scopul final al proiectului este acela de recunoaștere ale stărilor emoționale, de percepere și interpretare a trăsaturilor umane pe baza unor poze. Această recunoaștere se va putea face atât din pozele realizate cu camera web cât și din cele realizate cu telefonul personal.

## ***1.3Prezentarea pe scurt a capitolelor existente în lucrare***

În continuare, vor fi prezentate capitolele care vor face parte din această lucrare și prin care voi incerca să lămuresc anumite probleme referitoare la acest subiect. Lucrarea va fi împărțită, așadar, în mai multe capitole și subcapitole. Pentru început voi spune câteva cuvinte despre fiecare în parte, urmând ca descrierea în detaliu să fie făcută în cadrul fiecărui capitol.

Capitolele și subcapitolele prezente în lucrare:

**− Studiul bibliografic și Fundamentare teoretică** − în care va fi expus suportul teoretic care va sta la baza aplicației pe care doresc să o realizez; vor fi descrise limbajele de programare necesare implementării. Vor fi prezentate, în detaliu limbajele și mediile de programare Qml, Qt, Python, PyQt, motivele pentru care am ales sa folosesc aceste limbaje și nu altele, comparații între limbaje(asemănari si deosebiri).

− **Specificațiile aplicației**–în acest capitol se va prezenta schema bloc care stă la baza realizării acestei aplicații.Vor fi prezentate componentele acestei scheme, precum și relațiile dintre acestea. Tot în acest capitol, voi vorbi despre modul în care am gândit aplicația, despre interfața aplicației, care trebuie să fie una funcționala, simplă, pe care utilizatorul să o poată folosi fără prea multe probleme.

Capitolul **Specificațiile aplicației** va avea mai multe subcapitole:

− **Funcțiile sistemlui**− reprezintă de fapt o descriere a aplicației, o prezentare a ceea ce este ea și ce face;despre funcțiile generale ale sistemului.

− **Interfața cu utilizatorul − e**xplicarea pe scurt a folosirii interfeței de către utilizator.

− **Structura de baze de date si fișiere −** prezentarea fișierelor care trebuie convertite.!!!

După capitolul în cadrul căruia sunt prezentate specificațiile, urmează capitolele:

− **Proiectarea de detaliu −** reluarea capitolului anterior și explicarea detaliată a acestuia; se reia fiecare subcapitol și se insistă mai mult pe anumite părți.

− **Utilizarea sistemului −** va fi prezentat un manual de utilizare al aplicației pentru cei care vor folosi aplicația, se prezintă detaliat modul de utilizare pas cu pas al aplicației prin expunerea unor capturi de ecran.

− **Realizarea, punerea în funcțiune și rezultatele experimentale −** prezentarea rezultatului final, compararea rezultatului cu ipoteza, prezentarea impactului asupra utilizatorului, problemele întampinate la proiectare sau la utilizarea aplicației, rezolvarea acestor probleme.

− **Concluzii –** rezultate obținute, atingerea sau nu a scopului propus.

− **Referințe bibliografice** − menționarea bibliografiei utilizate pentru realizarea lucrării.

# **2. Fundamentare teoretică**

În cadrul acestui capitol voi prezenta fundamentul teoretic care va sta la baza realizării acestei lucrări. Voi prezenta mecanismele pe care le voi folosi în implementare, dar și explicații legate de acestea, voi argumenta de ce am ales aceste mecanisme și nu altele folosindu-mă de comparații între aceste mecanisme și alte mecanisme nefolosite din anumite motive, care desigur le voi preciza si explica în detaliu.

## ***2.1 Scurtă prezentare a mecanismelor folosite în implementare***

Mecanismele pe care le voi folosi sunt următoarele:

* OpenCV este o biblioteca de Python creată pentru a rezolva problemele de *compuuter vision*. Unul dintre principalele avantaje ale OpenCV este faptul că este foarte optimizat și disponibil pe aproape toate platformele.
* Haarcascade training ....
* Python este un limbaj ușor de folosit, oferă posibilitatea unei mai bune structurări, este un limbaj de nivel foarte înalt, dispune de tipuri de date de nivel înalt cum ar ﬁ tablouri sau dicționare foarte ﬂexibile, a căror implementare în C ar lua programatorului zile[2], îl voi folosi tocmai din aceste motive.
* *XML* (*Extensible Markup Language*) este un limbaj simplu care definește reguli de codificare ale documentelor într-un format lizibil omului
* LXML – librărie Python pentru citire si scriere XML-uri
* Android este un sistem de operare pentru telefoane mobile.
* Kivy

## ***2.2 Biblioteca OpenCV***

Aplicația realizată de mine se încadrează în domeniul numit *affective* *computing* , care permite dezvoltarea de sisteme și dispozitive care pot recunoaște, interpreta, procesa, și simula emoțiile umane . Acest domeniu este unul interdisciplinar care acoperă mai multe științe cum ar fi: știința calculatoarelor, psihologie și știintele cognitive. [9]

O motivație pentru cercetarea acestui domeniu o reprezintă abilitatea de a simula empatie. Mașina trebuie să interpreteze starea emoțională a oamenilor și să se adapteze comportamentului lor, dând un răspuns adecvat pentru acele emoții[10].

Pentru detectarea informației emoționale avem nevoie de senzori care captează date despre starea fizică a utilizatorului sau despre comportamentul acestuia fără a interpreta și datele de intrare. Un astfel de exemplu este camera video care poate capta expresii faciale, poziția corpului și gesturi. Alți senzori pot detecta semnalele emoționale cum ar fi temperatura pielii.

Identificarea informațiilor emoționale necesită extragerea modelelor semnificative din datele colectate. Acest lucru se face folosind tehnici de invățare care procesează diferite modalități de: reunoaștere a vocii , procesare a limbajului natural , sau de detectare a expresiilor.

În știintele cognitive există două modele de conducere care descriu modul în care oamenii percep și clasifică emoțiile: modelul continuu și modelul categoric. Modelul continuu definește fiecare expresie facială ca un vector caracteristic într-un spațiu al feței [9].

Modelele: categoric și continuu nu pot identifica mai multe emoții. Majoritatea sistemelor actuale sunt dependente de date, iar pentru detectarea emoțiilor bazate pe vorbire, sau detectarea celor bazate pe trăsaturile faciale este nevoie de baze de date adecvate folosite pentru a instrui clasificatorul[9].

Pentru detectarea și prelucrarea expresiilor faciale se folosesc diverse metode , cum ar fi: fluxul optic , modelul Markov ascuns și procesarea de rețele neuronale.

[Paul Ekman](https://translate.googleusercontent.com/translate_c?depth=1&hl=ro&prev=search&rurl=translate.google.ro&sl=en&u=https://en.wikipedia.org/wiki/Paul_Ekman&usg=ALkJrhhS9FkezyJr49nBNh2zIdFYMWQ7Xg)  a definit șase emoții de bază: furie, dezgust, frică, fericire, tristețe,surprindere. Pe baza acestora, a fost conceput un sistem pentru clasificarea și exprimarea fizică a emoțiilor. Acest sistem poartă numele de *FACS* ( creat de Paul Ekman și Wallace V. Friesen în 1978)

Pentru detectarea și prelucrarea facială, este necesară depășirea obstacolelor pentru evidențierea algoritmului și a metodei folosite. Pentru  asta se folosește curba lui Gauss , filtrare mediană,  sau metode mai noi.

Este cunoscut faptul că gradul de acuratețe în recunoașterea facială nu a fost adus la un nivel suficient de ridicat pentru a permite utilizarea lor pe scară largă(au existat multe încercări, dar nu s-a reușit identificarea cu success a infractorilor). Fără a îmbunătăți precizia de hardware și software folosită pentru a scana fețele, progresul este foarte mult încetinit[9].

Gesturile pot fi utilizate în mod eficient pentru detectarea unei anumite stări emoționale a utilizatorului, în special atunci când este utilizat în conjuncție cu vorbire și recunoaștere a feței. În funcție de acțiunea specifică, gesturile ar putea fi simple reflexe ( ridicarea umerilor atunci când nuse cunoaște răspunsul la o întrebare), sau ar putea fi complexe și pline de înțeles atunci când comunică cu limbajul semnelor. În cazul în care se folosesc obiecte, care pot fi atinse sau mutate, acestea ar trebui să fi și recunoscute, analizate și utilizate eficient de către un computer.

Detectarea stării emoționale a unui utilizator se poate face prin monitorizarea și analiza semnelor fiziologice. Cele trei semne fiziologice principale care pot fi analizate sunt: ​​pulsul volumului sanguin , răspunsul galvanic al pielii , electromiografie facială[9].

Acest domeniu de cercetare este încă în fază incipientă, dar care prinde avânt. Affective computing folosește diverse modalități pentru analizarea emoțiilor uman. Una dintre cele mai importante modalități este *Computer Vision*.

*Computer Vision este un* domeniu care include metode pentru dobândirea , prelucrarea, analizarea și înțelegerea imaginilor și, în general, date high-dimensionale din lumea reală , în scopul de a produce informații numerice sau simbolice. O temă dezvoltată în acest domeniu a fost dublarea capacitățile vederii umane , prin perceperea și înțelegerea unei imagini în format electronic, transformarea imaginilor vizuale în descrieri ale lumii și interfațarea cu alte procese. Această înțelegere a imaginilor poate fi văzută ca despletirea informației simbolice în *date image* folosind modele construite cu ajutorul geometriei, fizicii, statistici, și teoria învățării. *Computer Vison* a fost de asemenea descris ca automatizarea și integrarea unei largi game de procese și reprezentări pentru percepția vederii[11].

*Computer Vison* studiază teoria din spatele sistemelor artificial care extrag informațiile din imagini. *Image date* poate lua diverse forme: secvențe video, date multi-dimensionale de la un scanner medical. Această disciplină, caută să aplice teoriile și modelele sale în construirea de aplicații *Computer Vision*.

Inteligența artificială și *Computer Vision* împărtășesc aceeași viziune despre subiecte cum ar fi:  recunoașterea formelor și a tehnicilor de învățare. În consecință, *Computer Vison* este uneori văzută ca o parte a domeniului inteligenței artificial, dar și a domeniului *Affective Computing*[11].

*Computer Vision* este opusul graficii pe calculator. În timp ce grafica computerizata produce *image date* de la modele 3D, *Computer Vision* produce modele 3D de la *image date*.

O problemă des întâlnită în Computer Vision este aceea de a procesa imaginile, dar și de a determina dacă imaginile conțin anumite obiecte. Diferite varietăți ale problemei recunoașterii sunt descrise în literatura de specialitate:

* **Recunoașterea obiectului** (**clasificarea obiectelor)** - una sau mai multe obiecte sau clase de obiecte pot fi recunoscute, de obicei , împreună cu pozițiile lor 2D în imagine sau. *Blippar*, *LikeThat* furnizează programe independente care ilustrează această funcționalitate.
* **Identificarea** - o instanță individuală a unui obiect de recunoscut. Exemplele includ identificarea feței sau a amprentelor digitale al unei anumite persoane, identificarea cifrelor scrise de mână , sau identificarea unui anumit vehicul.
* **Detecție** – *image date* sunt scanate pentru o anumită afecțiune. Exemplele includ detectarea posibilelor celule sau țesuturi, sau detectarea unui vehicul într-un sistem automat de taxare rutieră. Detecția bazată pe calcule relativ simple și rapide este uneori folosită pentru identificarea unei regiuni mai mici de date a unei imagini[11].

Cei mai buni algoritmi pentru astfel de sarcini sunt cei bazați pe rețele neuronale convoluționale . Performanța acestor rețele pe testele ImageNet, este aproape de cea a oamenilor.

*Computer Vison* este extrem de dependent de aplicație. Unele sisteme sunt stand-alone. altele constituie un sub-sistem al model mai mare, care poate conține alte sub-sisteme pentru controlul dispozitivelor de acționare mecanice, planificare,etc.

O imagine digitală din cadrul unei aplicații din acest domeniu este produsă de unul sau mai mulți senzori de imagine, care, pe lângă diferite tipuri de camere sensibile la lumină, includ senzori cu rază, dispozitive de tomografie, radar, camere video ultra-sonic, etc. , în funcție de tipul de senzor, rezultând o imagine obișnuită 2D, un volum 3D, sau o secvență de imagini. Valorile pixelilor corespund uneia sau mai multe benzi spectrale (imagini gri sau imagini color)[11]. Este necesară pre-procesarea imaginilor/datelor în scopul îndeplinirii anumitor ipoteze implicite. Câteva exemple ar putea fi:

* + Re-eșantionare pentru a se asigura că sistemul de coordonate al imaginii este corectă.
  + Reducerea zgomotului pentru a se asigura că zgomotul senzorului nu introduce informații false.
  + îmbunătățirea contrastului pentru a se asigura că informațiile relevante pot fi detectate.

În procesul de prelucrare se iau o serie de decizii cu privire la punctele si regiunile cele mai relevante pentru o prelucrare ulterioară. Exemple:

* + Selecția unui anumit set de puncte de interes
  + Segmentarea uneia sau a mai multor regiuni de imagini care conțin un anumit obiect de interes
  + Segmentarea imaginii în scene formate din grupuri de obiecte, obiecte singulare sau părți ale obiectelor importante[11].

După pre-procesare, urmează procesarea la nivel înalt. În cadrul acestei se folosește un set mic de date de intrare , un set de puncte sau o regiune a unei imagini care se presupune că, conține un anumit obiect. Exemple:

* + Verificarea că datele satisfac modelele de bază.
  + Estimarea parametrilor specifici de aplicare, mărimea obiectului.
  + De recunoaștere a imaginilor - clasificarea unui obiect detectat în diferite categorii.
  + Înregistrarea imaginii - compararea și combinarea a două puncte de vedere diferite ale aceluiași obiect.

Luarea deciziei finale, de exemplu:

* + Pass / fail privind cererile de inspecție automate
  + Meci / no-meci în aplicații de recunoaștere
  + Pavilion pentru examinare umană și mai mult în aplicații medicale, militare, de securitate și de recunoaștereă[11].

Opencv

Aplicațiile Computer Vision sunt interesante și utile, dar algoritmii care stau la baza lor sunt intesinvi din punct de vedere computațional. Odată cu apariția de cloud computingavem mai multă putere de procesare în lucrul cu aceștia. Biblioteca OpenCV permite rularea algoritmilor Computer Vision în timp real. OpenCV a devenit biblioteca standard în acest domeniu. Unul dintre principalele avantaje ale OpenCV este faptul că este foarte optimizat și disponibil pe aproape toate platformele.

OpenCV este una dintre cele mai populare biblioteci utilizate pentru a dezvolta aplicații de tipul *Vision Computer*. Ea ne permite rularea mai multor algoritmi Vision Computer în timp real. Acesta a devenit biblioteca standard în acest domeniu. Unul dintre principalele avantaje ale *OpenCV* este faptul că este foarte optimizat și disponibil pe aproape toate platformele.

## ***2.3 Haarcascade training***

## ***2.4 Limbajul Python***

Acest limbaj este cel mai potrivit pentru elaborarea de *script*-uri, precum și pentru dezvoltarea de aplicații în mai multe domenii, pe majoritatea platformelor.

Este un limbaj des utilizat deoarece este ușor de învățat, dar este în același timp și un limbaj de programare puternic. Are nenumărate avantaje: oferă posibilitatea unei mai bune structurări, reprezintă un bun suport pentru programe mai mari.

Pe de altă parte dispune de o mult mai bună verificare a erorilor decât C-ul, și fiind un limbaj de nivel foarte înalt, dispune de tipuri de date de nivel înalt cum ar fi tablouri sau dicționare foarte flexibile, a căror implementare în C ar lua programatorului zile. Datorită caracterului general al tipurilor de date Python poate fi aplicat într-o arie mult mai largă de probleme decât AWK sau PERL. Cu toate astea, multe lucruri se realizează la fel de ușor în Python ca și în aceste limbaje[1].

Este foarte util deoarece permite divizarea programelor în module care pot fi folosite apoi în alte programe Python.

Fiind un limbaj interpretat, acesta accelerează munca în timpul procesului de dezvoltare, nu este nevoie ca de fiecare dată să se facă o compilare a codului.

Limbajul *Python* permite scrierea unor programe foarte compacte¸ ușor de citit. Programele scrise în Python sunt mult mai mici decât echivalentele în C sau C++ pentru că:

− **tipurile de date de nivel înalt** permit realizarea operațiilor complexe într-o singură instrucțiune;

−**gruparea instrucțiunilor** este dată de paragrafare în loc de blocuri begin/end sau de acolade;

−**nu este necesară declararea variabilelor**.

Python este un limbaj extensibil: dacă aveți cunoștințe de C puteți foarte ușor sa adăugați o nouă funcție sau un modul, fie pentru a efectua operații la viteză maximă, fie pentru a lega programe Python la biblioteci disponibile numai în formă binară (biblioteci grafice).[1]

### ***2.4.1 Structuri de control în Python***

Python dispune de structurile de control obișnuite, întâlnite și încelelalte limbaje.Acestea sunt:

− **instrucțiuni *if*** (instrucțiunea prezentă în toate limbajele de programare).

Fără această instrucțiune, nu s-ar putea face verificarea uneia sau mai multor condiții. În cadrul unui program pot exista mai multe secțiuni *elif* sau niciuna, iar secțiunea *else* este opțională. Cuvântul cheie “*elif”*  fiind prescurtarea de la ”*elseif”*, acesta este folositor pentru a evita tabularea excesivă. O secvență *if..elif..elif* funcționează ca un bloc case sau *switch*, secvențe proprii altor limbaje de programare [1].

− **instrucțiuni *for***

Instrucțiunile *for* din *Python* sunt diferite. Iterațiile instrucțiunii *for* din *Python* funcționează după elementele listei sau a unui șir, pe când în alte limbaje de programare, de exemplu în limbajul *C* iterația este una flexibilă, programatorul putând defini atât pasul cât și condiția de oprire.

− **funcția *range()***

Această funcție este folosită pentru generarea unor liste ce conțin progresii aritmetice. Parametrul furnizat funcției nu va fi niciodată un membru al secvenței. Secvența generată de această funcție să nu înceapă de la 0. Rația progresiei aritmetice poate fi modificată de fiecare dată.

**− instrucțiuni *break***

Instrucțiunea din cadrul acestui limbaj are aceeași semnificație ca și în limbajul de programare *C*, adică terminarea forțată a unei bucle *while* sau *for.*

Instrucțiunile din cadrul unei clauze else sunt executate atunci când bucla se termină odată cu terminarea listei (*for*) sau atunci când condiția buclei devine falsă (pentru *while*) aceste instrucțiuni nu sunt executate dacă bucla este terminată printr-o instrucțiune *break.*[1]

**− instrucțiuni *continue***

Această instrucțiune trece necondiționat la următoarea iterație.

**− instrucțiuni *pass***

Instrucțiunea “*pass*” poate fi folosită atunci când este necesară prezența sintactică a unei instrucțiuni, fără ca aceasta să execute ceva.[1]

**− instrucțiuni *funcții***

Cuvântul *def* este un cuvânt cheie în Python, este primul cuvânt folosit în definiția unei funcții. Acesta este urmat de numele funcției și de o listă de parametrii ai respectivei funcții.

Instrucțiunile care formează funcția încep pe linia următoare, și sunt plasate la o distanță diferită de margine față de definiția funcției.

Prima instrucțiune din corpul funcției poate fi un șir decaractere (opțional), acest șir reprezentând documentația funcției. În momentul în care se produce execuția unei funcții,se creează o nouă tabelă de simboluri, care se folosește pentru variabilele locale. Toate atribuirile din cadrul unei funcții vor stoca valorile în tabela de simboluri locală. Atunci când interpretorul găsește un nume devariabilă, întâi caută în tabela de simboluri locală, apoi în cea globală, și în final în cea predefinită.Cu toate acestea variabilelor globale nu li se pot atribui valori în cadrul unei funcții, decât dacă se folosește instrucțiunea global.

Parametrii actuali ai unei funcții apelate sunt introduși în tabela de simboluri locală a acelei funcții în momentul apelăriiei. Transferul argumentelor făcându-se prin utilizarea unui apel prin valoare(unde valoarea este întotdeauna un obiect referință, nu valoarea obiectului).Când o funcție apelează o alta funcție, este creată o nouă tabelă de simboluri pentru funcția apelată.

Definirea unei funcții trece numele funcției în tabela de simboluri curentă.[1]

**− Liste în *Python***

Acest tip de date dispune de câteva metode. Acestea sunt:

**•** *append(x) −*Adăugarea unui element la sfârșitul listei.

**•** *extend(L) −*Adăugarea elementelor furnizate ca și parametru la sfârșitul listei.

**•** *insert(i, x) −*Inserarea unui element într-o anumită poziție. Primul argument reprezintă indicele elementului dinlistă înaintea căruia se va face inserția.

**•** *remove(x)* −Ștergerea din listă a primului element găsit cu valoarea ’x’. Dacă nu există un astfel de element apare o eroare.

**•***pop([i])* − Ștergerea din listă a elementului de pe poziția ’i’, și returnarea valorii acestuia.

**•** *index(x) −* Întoarce indicele primului parametru din listă care are valoarea ’x’.

**•***count(x) −* Întoarce numărul de apariții ale valorii ’x’ între elementele listei.

**•***sort() −* Sortează elementele listei.

**•** *reverse() −* Schimbă ordinea elementelor din listă[1].

Din cauza organizării elementelor în cadrul fișierelor *Xml* pe care doresc să le convertesc într-un format ușor de vizualizat, voi folosi atât liste cât și dicționare; așadar parcurgerea se va face în funcție de tipul de date prezent în fișiere. În momentul conversiei se va ține cont de aceste structuri de date, conversiile fiind diferite, în sensul că, dacă un element este de tip listă va arăta diferit față de cel care inițial era de tip dicționar.

**−Dicționare**

La fel ca și lista, dicționarul reprezintă un tip de date predefinit, pe care îl voi folosi în implementare, fiind foarte folositor. Se pot întâlni și în alte limbaje de programare dicționare, dar sub o altă denumire: “memorii asociative” sau “ valori asociative”.

Dicționarele sunt indexate de *chei*, carepot fi definite de oricare din tipurile de date invariabile(nemodificabile). Perechile pot fi folosite drept chei ale unui dicționar numai dacăconțin obiecte invariabile. Nu se pot folosi drept cheilistele deoarece acestea pot fi modificate folosind metode ca *append()* sau *extend()*.Este indicat săne gândim la un dicționar ca la o mulțimeneordonată de perechi cheie-valoare, cu observațiacă o cheietrebuie să fie unică într-un dicționar. O pereche de acolade creează un dicționar gol[1].

Operațiilerealizate de un dicționar sunt:

**•** stocarea uneivalori cu anumită cheie

**•**  extragerea unei valori cunoscându-se o anumită cheie.

O pereche cheie-valoare poate fiștearsă folosind instrucțiunea*del*. Dacă se adaugă o valoare în dicționare pentru o cheie care există deja, valoareaveche asociată acelei chei este pierdută.

Metoda *keys()* a unui obiect dicționar întoarce o listă cu toate cheile existente în respectivul dicționar. Lista returnatănu este sortată, însă pentru a o sorta se poate folosi metoda sort() a obiectului listă returnat de funcție.

## ***2.5 Limbajul XML***

Fișierele folosite in detectarea trăsaturilor umane sunt reprezentate de o serie de coordonate și au formatul XML, din această cauză voi incerca să descriu pe scurt acest limbaj, pentru ca oricine va vedea fișierele în acest format să poată să înțeleagă elementele de bază, să poată să-și facă o idee referitor la ce se află în acel fișier, chiar dacă până acum nu a avut de-a face cu acest limbaj.

Limbajul XML, acronimul de la *eXtensible Markup Language*, a fost definit de consorţiul W3C în anul 1996.

Voi prezenta în câteva cuvinte XML-ul:

* la fel ca şi HTML se bazează pe *tag*-uri.
* XML poate fi folosit de autorii de pagini Web ale căror nevoi depăşesc limitele HTML-ului.
* XML a fost creat pentru descrierea structurilor de date
* Tag-urile XML nu sunt predefinite; altfel spus, XML permite crearea tag-urilor personalizate, ceea ce este nemaipomenit.
* Pentru descrierea structurilor de date, documentele XML pot utiliza un DTD (Document Type Definition).
* Un document care conţine un DTD este autodescriptiv.

*Xml* -ul este un metalimbaj utilizat pentru descrierea datelor.[3]

Fie că se dorește crearea unui document *Xml* sau doar studierea și parcurgerea unui document *Xml* trebuie ținut cont de faptul că acest metalimbaj are niște reguli care trebuie respectate. Acestea reguli pot fi: de sintaxă *Xml* și de validitate a documentelor *Xml*.

Reguli de sintaxă *Xml:*

**− Închiderea tuturor elementelor −** toate elementele care se găsesc în cadrul unui document *Xml* trebuie să aibă un tag de deschidere și unul de închidere.

**− Închiderea tuturor elementelor vide −** dacă în cadrul unui document *Xml* există unul sau mai multe elemente vide, acestea nu pot fi lăsate deschise, trebuie închise.

**− Utilizarea ghilimelelor pentru a delimita valorile unor atribute −** dacă se folosesc atribute, valorile acestora trebuie neapărat puse între ghilimele.

**− Toate atributele folosite într-un document *Xml* trebuie să conțină valori −** fiecare atribut trebuie să aibă atribuită o valoare.

**− Elementele *Xml* sunt *case-sensitive* −** spre deosebire de alte limbaje, limbajul *Xml* este unul *case-sensitive*, literele mici fiind interpretate diferit față de cele mari. Așadar elementele dar și atributele trebuie utilizate așa cum au fost ele descrise în *DTD* –ul (*Document Type Definition*) asociat. În cazul în care nu se folosește un *DTD*, trebuie avut grijă ca fiecare *tag* de deschidere să aibă un *tag* de închidere.

**− Imbricarea corectă a elementelor −** un element poate conține și alte elemente, asta înseamnă că elementele se pot imbrica. Există și niște reguli referitoare la modul de imbricare și anume:

− Fiecărui tag de deschidere trebuie să-i corespundă un tag de închidere. Numele celor două tag-uri trebuie să fie identice.

− Tag-ul de închidere al unui element “fiu” trebuie să preceadă tag-ul de închidere al „părintelui” său. Altfel spus, un element „fiu” trebuie să fie închis înaintea „părintelui” său.

− Elementele imbricate sunt „fii ai elementului părinte”[3].

Entitățile analizate recursiv sunt interzise, adică un element nu se poate referi la el însuși.

Un document *Xml* este un document “bine format” dacă respectă anumite reguli structurale și notații *Xml*.

Reguli de validitate a documentelor *Xml* :

**− Utilizarea elementului rădăcină −** orice document *Xml* trebuie să conțină cel puțin un element rădăcină, iar acest element trebuie să le conțină pe celelalte elemente ale documentului.Un element rădăcină este delimitat printr-un tag de deschidere plasat la începutul documentului şi un tag de închidere plasat la finele documentului[3].

**− Un document*Xml* este considerat valid dacă conține un *DTD (Document Type Definition)* − a**șadar un document *Xml* este format din două părți:

**− Prolog (declarația documentului)**

**− Elementul rădăcină**

După această scurtă introducere în *Xml* și însușirea unor cunoștințe de bază în acest limbaj, putem să privim altfel fișierele cu care avem de-a face.

### ***2.5.1 Scheme XML prezente în lucrare***

Formatul fișierelor este așadar unul bazat pe XML, fiind construit cu ajutorul XML Schemă(XSD), vor fi prezentate și astfel de scheme pentru o mai bună înțelegere.

Schema XML este un limbaj de descriere pentru fișiere XML, permite definirea structurii și a conținutului ale unor astfel de fișiere, ceea ce înseamnă o definire gramaticală. Aceasta poate fi folosită pentru a exprima un set de reguli la care un document XML trebuie să se conformeze, în scopul de a fi considerat "valabil", în conformitate cu schema.

Schema XML este apoi utilizată pentru a defini meta-modelulXML-PSA și pentru a valida documentul rezultat din aceste meta-modele. Extensia asociată cu XML-PSA este PXA.

Meta-modelul XML-PSA definește un set de elemente XML care poate fi utilizat la crearea unui fișier PXA.

În cadrul modelului sunt prezente 3 familii principale de obiecte:

• Obiecte grafice 2D

• Obiecte grafice 3D

• Animații

În PXA Schema ,“*Object*”este un obiect non grafic. Acesta este cel mai mare obiect din diagrama de clase.

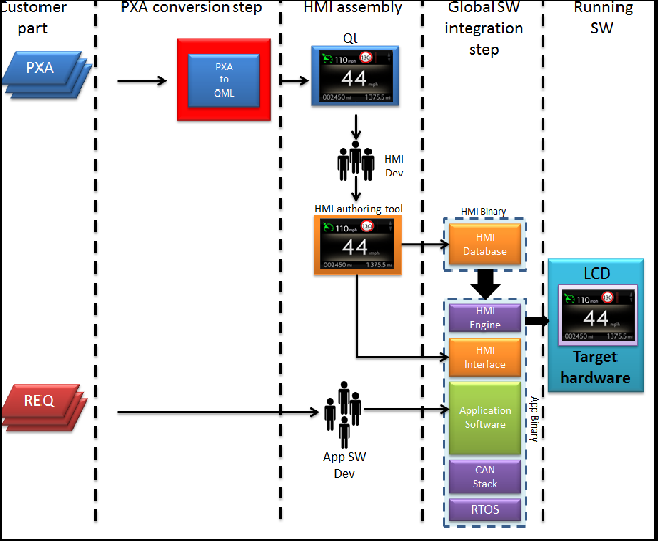
# 

# **3. Specificațiile aplicației**

## ***3.1 Schema bloc a sistemului. Scurtă descriere a aplicației***

Aplicația constă în realizarea unui convertor capabil să convertească fișierele primite în format *Xml* în fișiere *Qml*. În aceste fișiere se găsește cod de *Xml* prin care este descrisă partea grafică a specificațiilor. Din cauza faptului că trebuie redată starea specificațiilor și în formă vizuală (pentru a putea fi implementată partea de grafică), aceste fișiere de *Xml* cu extensia *pxa* trebuie importate într-un program care să simuleze *HMI*-ul pe un calculator.

La baza oricărei aplicații stă o schemă logică, în care sunt prezentate componentele de bază ale aplicației, precum și legăturile dintre ele. Voi prezenta schema care stă la baza aplicației mele, dar și schema care stă la baza întregului proiect, încercând să evidențiez prin aceasta partea care îmi este atribuita mie, prezentând și explicând fiecare modul în parte, legătura dintre ele, importanța fiecărui modul din proiect și mai ales importanța modului/aplicației mele.Schema generală a proiectului este prezentată în Fig 3.1.1.



*Fig 3.1.1 Schema generală a proiectului*

Voi incerca să explic pe larg toate modulele, legăturile dintre ele și cum în final, acestea puse cap la cap o să producă un anumit rezultat, rezultatul trebuind să fie desigur cel așteptat, conform cerințelor clientului.

Așadar, întreg proiectul se împarte in 5 module, unele mai complexe, altele mai puțin complexe si anume:

Modulul ***Customer part***reprezintă partea de specificații primite de la client; acestea sunt împărțite în două:

− partea de **specificații scrise**–specificații la nivel funcțional ale tuturor elementelor HMI descrise de PXA; în acestea sunt descrise clar cerințele și așteptările pe care aceștia le au de la companie. Sunt diferite specificații,pentru fiecare domeniu in parte, astfel pot sa fie specificații pentru partea de *software,* de *hardware*, de *mecanica*, de *calitate*, chiar și de *producție.* Acestea sunt foarte importante pentru ducerea la bun sfârșit a proiectului, dar și pentru obținerea rezultatului dorit. La aceste documente au acces toți angajații, fiind prezente într-un *tool*la care are acces toata lumea. Prezintă importanțăși felul in care se prezintă aceste specificații primite de la client, în sensul că pot face lucrurile mai grele sau mai ușoare; dacă sunt neclare și foarte greu de perceput o sa apară probleme la implementare, înțelegându-se cu totul altceva, rezultatul poate să nu fie cel așteptat, ca urmare a specificațiilor și nu a implementării. Aceasta este o problema gravă, deoarece se pierde foarte mult timp cu rezolvarea ei, fiind nevoie de verificări amănunțite legate de toate modulele și funcțiile în parte pană la constatarea problemei specificațiilor; odată descoperită această problemă, trebuie rezolvată prin verificarea specificațiilor de către cei care le-au creat și rezolvarea problemei: fie prin modificarea lor în așa fel încât să fie clare și să poată fi înțelese de toată lumea, fie în cel mai rău caz refacerea lor. Ambele variante necesită timp și bani, aceștia fiind cei mai importanți factori in realizarea unui proiect.

−***pxa*-urile −** reprezintă cealaltă parte de specificații și anume specificațiile grafice, aici apar imagini care în final ar trebui sa apară pe *display.* Sunt expuse pentru fiecare modul in parte specificații legate de partea grafică, dar există o problemă: așa cum am menționat și în primul capitol (cel de **Introducere**), aceste imagini nu sunt vizibile, în sensul că nu pot fi văzute, pentru că aceste fișiere de descriere care conțin cod *Xml* nu pot fi deschise, neexistând un program cu care să se poată deschide fișiere cu extensia pe care acestea o au(*pxa*), din acest motiv trebuie creat un „program” cu ajutorul căruia să se rezolve aceasta problemă. Realizarea acestui minus prezintă o deosebită importanță pentru finalizarea proiectului, deoarece va fi folosit de către cei care se ocupă de implementarea părții grafice, aceștia având nevoie să poată vedea imaginile pe care urmează să le prelucreze și săle poată afișa pe *display* . Acest modul implică existența celui de-al doilea modul și anume ”Convertorul Pxa”.

**Convertorul Pxa**reprezintă modulul realizat de mine. Pana acum am încercat să explic de ce este nevoie de un asemenea modul și am subliniat în nenumărate rânduri importanța sa. Fără de acest modul nu s-ar putea continua proiectul, deci nu ar ajunge prea curând la finalizare, lucru deloc indicat. Acest convertor va converti fișierele date în format Xml și cu extensia *pxa*(primite de la client)în fișiere *qml* care se pot deschide pentru vizualizarea conținutului acestora. Schema care stă la baza realizării acestui convertor va fi prezentată mai jos, însoțită de explicații privind componentele schemei, dar și legătura dintre acestea.

Acest modul este în strânsa legătura cu următorul modul(Asamblare HMI), fără acest modul, modulul de asamblare nu s-ar putea realiza.

**Asamblare HMI**este un modulul în care se folosescfișierele convertite cu ajutorul convertorului pxa. Cei desemnați să se ocupe de partea graficăiau fișierele *pxa* de care au nevoie și introduc în cadrul aplicației calea către fișierul pe care doresc să-l convertească, astfel fișierele dorite vor fi convertite și vor putea fi vizualizat. Odată realizată această problemă, se poate trece la următoarea etapă, adică adăugarea imaginilor obținute într-un *tool* pentru a putea fi prelucrate.

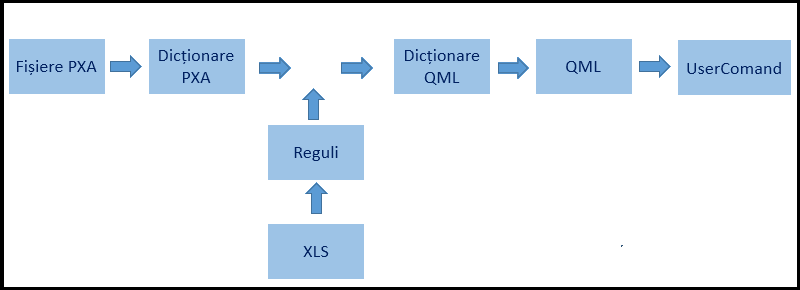
Tot in acest modul se realizează și implementarea specificațiilor scrise ale proiectului. Pentru aceasta se analizează fiecare specificație în parte, după care se realizează codul care trebuie să reflecte cele precizate in specificație.

După realizarea acestor module mai mici, se va trece la următorul modul în care se vor integra(“se vor pune cap la cap”) cele realizate pană acum, scopul fiind acela de a vedea dacăaceste mici module funcționează împreună.

Modulul de **Integrare globală** reprezintămodulul în cadrul căruia se integrează toate modulele mai mici la care s-a lucrat pana acum, adică partea de HMI *binary*se va integra în modulul de aplicație denumit *App binary*adevărata aplicație care va rula pe t*arget*-ul real. Această aplicație este formată din alte module mai mici care odată integrate ar trebui să funcționeze împreună fără probleme. Modulele mai mici care, integrate formează acest modul sunt: *HMI engine*, *HMI interface*, *application software*, *Can stack*, *RTOS*.

Odatăfăcută integrarea, nu rămâne de făcut decât compilarea soft-ului și punerea acestuia pe target. Care ar trebui să fie rezultatul? Cel așteptat, desigur, adică afișarea elementelor dorite pe display, display care va apărea în bordul mașinii pentru care a fost creat.

După prezentarea schemei generale a proiectului, urmează prezentarea schemei aplicației realizate de mine sau a modului care în cadrul proiectului real poartă numele de „Convertor PXA”, așadar schema gândită pentru realizarea aplicației este următoarea:



*Fig 3.1.2 Convertor limbaj descriptiv HMI*

Așa cum reiese și din schema de mai sus(Fig 3.1.2) părţile componente sunt:

− blocul în care apar fișierele cu extensia *pxa* − care conțin cod *Xml* ; fișierele urmând a fi citite (trebuie parcurs codul de *Xml* şi citit) pentru a crea mai târziuniștedicționare de *Pxa* care conținniște reguli așa cum reies din prezentele fișiere;

− blocul de creare a dicționarelor Pxa − de acest bloc am nevoie deoarece vreau ca la un moment dat să fac o comparație între dicționarele de *pxa* şi cele de *qml* pentru a vedea asemănările şi deosebirile dintre ele şi pentru a găsi un punct comun, o legătură între ele, care să-mi permită să dezvolt aplicația şi să obțin în final rezultatul dorit.

− bloc XLS − reprezintă*Excel* generat care conţine regulile pentru dicționare, asocieri şi echivalențe între acestea.

− bloc generator de dicționar de reguli− este cel care creeazăun dicționar și îl exportă în *Excel,* de care am nevoie pentru o mai buna afișare şi ordonare a regulilor dicționarelor.

− bloc care conţine dicționarul de Qml –este creat şi un dicționar de qml, pentru a vedea ce conţine un fișier de *qml* şi pentru a face comparație între dicționarele de pxa şi cele de *qml*, pentru a ajunge la o concordanță între acestea şi pentru a putea în final să convertesc aceste fișiere de *pxa* în fișiere de *qml*.

− bloc care transformă fișierele de *qml* in fișiere*string* de *qml* − va afișa în final o interfață prietenoasă cu utilizatorul.

## ***3.2 Funcțiile sistemului***

Aplicația poate fi împărțită în mai multe părți, elementele care compun această aplicație le voi prezenta în cele ce urmează.

Primul element este reprezentat de **fișierele pxa** primite de la client, care reprezintă o parte a specificațiilor de natură grafică. Aceste fișiere au descrierea prezentată în *Xml* şi de asemenea au extensie pxa; acestea sunt intrările în sistem, sunt considerate intrări deoarece acestea reprezintă punctul de plecare şi prin prelucrarea acestora, precum şi prin adăugarea altor module o să poată fi realizate ieșirile/rezultatul, rezultat pe care la început am presupus ca îl voi obține.

Așa cum am mai amintit în această lucrare, formatul fișierelor este unul bazat pe *Xml*, și a fost construit cu ajutorul *Xml* Schemă(XSD). Schema originală fiind expusă în această lucrare cu numele de Fig 2.2.1.1. precum şi diferite obiecte, elemente dar şi arborele *pxa* în care sunt prezentate anumite componente.

O să prezint mai multe fișiere de pxa şi o să explic mai pe larg cum am procedat pentru a le parcurge şi pentru a le transforma in dicționare, regulile pe care le-am folosit şi cum m-am gândit la această soluție, cum am generat *xls*-urile, toate în capitolul în care o să explic proiectarea în detaliu.

Aceste fișiere cu format *Xml*le voi parcurge şi le voi transforma în dicționare, dar nu voi modifica conținutul acestora, adică nu voi modifica codul din interiorul fișierului, doar pentru a arata mai bine rezultatul final, scopul nu este acesta, scopul este acela de a obține un rezultat fără ca aceste fișiere primite de la client să sufere vreo modificare.

Ca în orice aplicație software (program construit special pentru a rezolva anumite cerințe, în cazul meu programul este realizat pentru a se putea vizualiza niște fișiere care nu au un format comun) există niște primi pași care trebuie urmați şi anume: pentru început trebuie studiată şi analizată problema, urmează apoi identificarea datelor de intrare precum şi a celor de ieșire, trebuie elaborat un anumit algoritm de rezolvare a problemei. Am respectat acești pași într-totul: am studiat foarte bine problema, am identificat datele de intrare care sunt, după cum am spus şi mai sus, chiar fișierele primite de la client. Elaborarea algoritmului a reprezentat un pas destul de greu deoarece problema nu a mai fost tratată până acum, fiind o noutate. Un următor pas (unul dintre cei mai importanți) în realizarea unei astfel de aplicații îl reprezintă scrierea programului propriu zis, este adevărat că fiecare pas are însemnătatea lui, dar acest pas vine ca o concretizare a pașilor anteriori.

Una dintre primele operații cu o însemnătate deosebită în momentul în care se scrie programul ( mai ales în lucrul cu fișiere) o reprezintă operația de citire.Așadar nu se poate face nimic până în momentul realizării operației de citire, odată realizată această operație totul devine mai simplu. Al doilea element important în această aplicație îl reprezintă **dicționarele de pxa**. De ce am creat asemenea dicționare? La ce le folosesc, de ce sunt importante ele? Ideea de la care am plecat este următoarea: cum în aplicația mea, pentru realizarea acestui convertor folosesc script de *Python* şi cum *Python*-ul are o colecție bogată de biblioteci şi lucrează cu liste ordonate, dicționare, cu care nu se poate lucra în orice limbaj de programare şi cum pentru a găsi o cale de mijloc între fișierele de pxa şi fișierele generale de *qml* trebuie mai întâi realizată o comparație, care nu se poate face altfel decât prin realizarea unor dicționare în *Python*. Dicționarele de *pxa* se creează după citirea fișierelor *Xml*.

Se generează apoi un dicționar de conversie care conține reguli de conversie, acest dicționar reprezintă un al treilea element important în cadrul acestei aplicații; elementele care apar în aceste fișiere şi echivalentul lor in *qml*. Trebuie făcut în așa fel încât fiecare element din *pxa* să aibă un element echivalent în *qml,* astfel că la convertirea în qml, elementul pxa să fie transformat fără probleme în echivalentul său in *qml*. Un fișier *pxa* complex conţine foarte multe elemente, astfel că şi dicționarul creat pe baza acestuia va fi unul vast şi cu foarte multe reguli, reguli (al treilea element) care ar trebui puse într-un *xls* în care să fie ordonate şi să fie puse după tip (daca elementul este de tip listă sau item sau orice altceva) specificându-se o serie de criterii.

Pentru a putea introduce într-un mod automat regulile aparținânddicționarului de conversie(elementele din *pxa* şi echivalentul lor în *qml* este necesar un fișier ), am nevoie de un al patrulea element prezentat mai jos.

Al patrulea element îl reprezintă *xls*-ul care a fost generat prin cod *Python* (în *Python*există o serie de biblioteci care pot fi folosite pentru a crea foarte multe funcționalități). Regulile scrise în *Python* pentru dicționarul de pxa au fost importate în xls într-un mod foarte ordonat, fiind create mai multe *sheet*-uri în care sunt introduse reguli: generice, atribute, enumerări; fiecare dintre aceste reguli mai pot fi împărțite şi în funcție de tip.

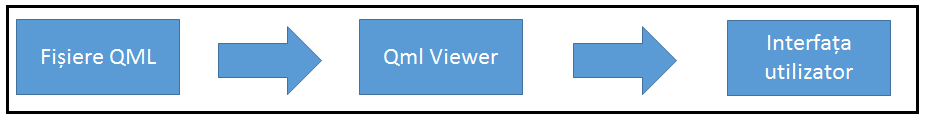
O acțiune importantă o reprezintă convertirea fișierelor pxa cu ajutorul dicționarelor create care conțin regulile despre care am discutat dar şi convertirea dicționarelor de *qml* in fișiere*string*de *qml*. Modulul de conversie este cel mai important din această aplicație, acesta este de fapt ce trebuia realizat, momentul în care necunoscutul devine cunoscut, momentul în care fișierele pxa vor putea fi deschise şi se va vedea ce elemente de *qml* conțin, iar prin crearea unei interfețe vor putea fi şi vizualizate de către oricine dorește, dar mai ales de către cei care vor modifica imaginile care apar în aceste fișiere.

Un ultim element în cadrul acestei aplicații îl reprezintă *usercmd*-ul care reprezintă de fapt un modul în cadrul căruia se realizează partea de interfață cu utilizatorul. Orice aplicație de tip *software* trebuie să aibă o interfață cu utilizatorul, de dorit una prietenoasă, iar acest modul reprezintă o rezolvare a acestei probleme.

## ***3.3 Interfața cu utilizatorul***

Cunoscând intrările și ieșirile din sistem, se poate proiecta o schemă bloc a interfeței cu utilizatorul. Așadar intrarea în blocul de intefață va fi reprezentat de fișierele Qml tocmai convertite; acestea vor fi adăugate in Qml Viewer. Pachetul UI include și Qt Qml Viewer, instrument care este folosit în general și pe care eu l-am folosi pentru a încărca fișierele qml pe care tocmai le-am convertit. În momentul in care acestea vor fi încărcate/adăugate, va apărea interfața creată de mine care conține o imagine (care inițial avea un format care nu se putea deschide), acum fiind accesibilă și putând fi deschisă, având format qml. Interfața mai conține și niște butoane cu ajutorul cărora se poate mări sau micșora imaginea, acesta reprezentând ieșirea blocului interfeței cu utilizatorul;

Schema bloc pentru interfață este următoarea:



*Fig 3.3.1 Schema bloc pentru interfață*

Interfața cu utilizatorul este una foarte simplă, pentru ca orice utilizator care nu are cunoștințe legate de programele în care s-a realizat aceasta să poată să o folosească cu un minim de cunoștințe, desigur voi prezenta și un manual de utilizare, ca să nu existe dubii legate de folosirea interfeței, iar imaginile care vor trebui modificate să poată fi accesate și vizualizate, totul pentru o bună desfășurare a lucrurilor. Manualul de utilizare precum și alte explicații, capturi cu interfața vor fi prezentate într-un alt capitol.

Așa cum am spus, pentru dezvoltarea interfeței cu utilizatorul folosesc *Qt* pentru că suportă diferite platforme și medii integrate (mediul integrat folosit de mine este *Qt Creator*), dar și pentru că interacționează cu limbajele de programare pe care eu le folosesc (*Python* și *Qml*). Mediul integrat folosit de mine include și un editor de cod, dar integrează si *Qt Designer*(instrument) pentru proiectarea și construirea de interfețe grafice*.* Fișierele de *qml* convertite pot fi vizualizate și în *Qt Designer.* Prin dublu *click* pe fișierele convertite, se va deschide editorul de *Qt*unde apare cod de qml (codul de *Xml*este convertit în cod de *qml,* iar în momentul în care se deschide editorul de *Qt*se poate observa rezultatul) echivalentul codului de *Xml*din fișierele inițiale de *pxa.* Odată văzut codul real de *qml*, se poate vedea și partea de grafică prin trecerea în *Qt Designer.*

## ***3.4 Structuri de date și fișiere***

### ***3.4.1 Fișiere***

Fișierele cu care lucrez sunt cele primite de la client, ele reprezintă o parte a specificațiilor grafice. Aceste fișiere conțin imagini care la început nu sunt vizibile; necesitatea afișării în final pe *display* a imaginilor conform specificațiilor implică realizarea programului care să fie capabil să afișeze într-o interfață cu utilizatorul ceea ce conțin de fapt fișierele care nu vor mai avea extensia *pxa,* ci vor avea extensia *qml.*

Voi prezenta mai jos niște modele de fișiere: un model primit de la client, explicând ce conţine de fapt acesta şi un model de fișier convertit (cum va arăta de fapt fișierele convertite, ce cod vor conţine şi care este semnificația acestuia).

<Panel >

<statesBindings>

<state name=” ” />

</statesBindings>

<data>

<Image id=” ” />

<ImageIndex id=” ” />

</data>

<states>

<State name=” ” />

</states>

<transitions>

<Transition from=” ” to=” ”>

<animations>

<NumberAnimation duration=” ” properties=” ”>

<targets>

<ref>

</ref>

</targets>

</NumberAnimation>

<SequentialAnimation>

<animations>

<PauseAnimation duration=” ” />

<NumberAnimation duration=” ” properties=” ”>

<targets>

<ref>

</ref>

</targets>

</NumberAnimation>

</animations>

</SequentialAnimation>

</animations>

</Transition>

</transitions>

<statesConditions>

<StateCondition when=” ” state=” ”>

</statesConditions>

</Panel>

După cum se poate observa, codul prezentat mai sus (apareîn fișierele primite), este un cod *Xml.* Pentru a înțelege structura fișierelor este necesar să se înțeleagă codul din interiorul fișierelor, din această cauză trebuie ca să fie înțelese în primul rând regulile acestui limbaj. Regulile sunt unele simple, putând fi: fie reguli de sintaxă, fie reguli de validitate a documentelor *Xml.* Nu voi insista asupra acestor reguli, deoarece le-am prezentat în cadrul capitolului denumit “Fundamentare teoretică”.

Orice document *Xml* ar trebui să respecte acele reguli*.* Una dintre cele mai importante este aceea că: fiecare document *Xml* trebuie să aibă cel puțin un nod rădăcină, acest nod trebuie să conțină celelalte noduri ale documentului. In cazul fișierelor primite este respectată această regulă. Nodul rădăcină care apare în majoritatea fișierelor cu format *Xml*este *Panel,* dar mai sunt şi excepții în care nodul rădăcină este denumit *Container* şi nu *Panel.*Așa cum se observă din modelul de mai sus, elementul rădăcină este delimitat printr-un tag de deschidere plasat la începutul fișierului şi un tag de închidere plasat la finalul documentului. Deși în modelul prezentat am pus doar nodurile care fac parte dintr-un fişier *Xml,* declarația*Xml*nu lipsește, deși ea este facultativă. În cadrul acestor fișiere, declarația acceptă ca şi atribut, atributul *encoding*"UTF-8", prin care se semnalează faptul că documentul este unul *Xml.* Versiunea folosită este 1.0,în fișier mai apar şi alte versiuni, care țin strict de *pxa.*

Fiecare element din cadrul acestor fișiere cuprinde: un tag de deschidere, conținutul şi un tag de închidere. Un element poate avea de asemenea şi alte elemente imbricate. În modelul expus, nodul rădăcină este nodul *Panel* care conţine următoarele elemente: *data*, *states*, *transitions*, şi*statesConditions.*

Elementul *data* conţine şi alte elemente imbricate cum ar fi: *Image,* care poate avea mai multe atribute: *id, partName, source*, dar şi altele, în funcție de modulul din care face parte fișierul. Desigur, există probabilitatea ca elementul *data* să conțină mai multe elemente imbricate, care să aibă același nume, evidențiate prin *id*.Atributele oferă informații despre imaginea care ar trebui să apară în final. De fapt toate elementele care compun fișierele reprezintă o descriere a imaginii care urmează să fie afișate. Este foarte important să fie precizate anumite caracteristici ale imaginii. Prin atributele enumerate mai sus şi folosite în fișiere se specifică de fapt un *id* care trebuie sa fie unic, numele imaginii şi calea unde se găsește.

Elementul *states* conţine elementul *State*cu atributul *name* se referă la starea unei imagini care poate să fie: *opening* sau *closing.*

Un alt element care apare în cadrul unui fișier*pxa*este reprezentat de elementul *transitions* care poate avea elemente imbricate, acestea la rândul lor pot să aibă alte elemente imbricate. Un element imbricat al elementului*transitions* este *Transition* care poate să aibă imbricat elementul *animations* cu elementele: *NumberAnimation* cu o serie de atribute: *duration*, *properties,*darşi altele;*SequentialAnimation* cu elementul *animations* care poate să aibă elementele: *PauseAnimation*şi *NumberAnimation* fiecare putând să mai aibă şi alte elemente imbricate, care poate să aibă unul sau mai multe atribute. În cadrul acestor elemente este descris modul în care se realizează tranzițiile.

Un ultim element care ar putea să apară este *statesConditions* cu elementul *StateCondition*care poate să conțină anumite condiții. Pot să mai apară şi alte elemente în cadrul fișierelor, dar elementele care se regăsesc în toate fișierele primite sunt elementele prezentate mai sus.

### ***3.4.2 Structuri de date***

Un program software este alcătuit din algoritmi(unul sau mai mulți) şi structuri de date. Operația de alegere a unei structuri de date reprezintă un obiectiv principal al specificațiilor de implementare. Calitatea și performanța produsului final depind de structura de date aleasă. Algoritmii utilizați sunt evidenți în momentul când sunt folosite structurile de date potrivite, însă dacă acestea sunt alese greșit, situația se complică. Unele limbaje permit reutilizarea structurilor de date. Din cauza faptului că structurile de date sunt atât de importante, ele pot fi incluse în bibliotecile standard ale limbajelor de programare precum și în mediile de dezvoltare. Acesta fiind unul dintre motivele pentru care am folosit mediile de dezvoltare și limbajele de programare menționate încă de la începutul lucrării. Limbajele folosite de mine conțin foarte multe biblioteci din cadrul cărora fac parte multe clase și obiecte fără de care implementarea ar fi mult mai greoaie.

Clasele, listele și toate elementele care le folosesc în cadrul implementării, le folosesc tocmai pentru construirea structurilor de date.

Structurile de date reprezintă implementări ale unor interfețe: o structură de date poate fi văzută ca o interfață între două funcții sau ca o implementare a metodelor de accesare a depozitului care este organizat în concordanță cu tipul de dată asociat[5].

O importanță deosebită o au obiectele și pachetele primite de la client, prin care se specifică modul în care se leagă elementele din schema *Xsd*descriindu-se astfel fișierele cu for mat *Xml*care în final sunt convertite în *qml*. Prin prezentarea acestor obiecte se creează o imagine destul de clară asupra modului în care trebuie tratată problema.

# **4. Proiectarea de detaliu**

## ***4.1 Arhitectura programului***

În cadrul unei arhitecturi software este descrisă structura unui sistem de programe la nivel abstract. Arhitectura programului meu, constă dintr-un set de componente, conectori şi configurații care sunt caracterizate de un anumit șablon(tip arhitectural). Tipul arhitectural trebuie ales în așa fel încât să fie unul potrivit, pentru ca arhitectura să fie în conformitate cu specificațiile. Trebuie așadar găsită o metodă prin care un astfel de program complex să poată fi evaluat din punct de vedere al fiabilității. Această metodă oferă un mijloc prin care proiectanții să poată proiecta arhitectura care se potrivește cel mai bine cu specificațiile primite de la client. În același mod am procedat şi eu, atunci când am creat arhitectura aplicației mele, am ținut cont de specificațiile primite de la client.

În cadrul schemei prezentate în Fig 3.1.2 sunt ilustrate componentele aplicației mele, precum şi conectorii. Voi descrie detaliat fiecare componentă în parte din cadrul acestei scheme, explicând relația dintre ele, precum şi modul în care au fost create.

Pentru crearea acestor componente am folosit ca şi medii de dezvoltare următoarele medii:*Qt* 5.1, *Eclipse Kepler* cu *plugin*-ul de *PyDev*în care am inclus extensia *PyQt4(*pentru limbajul *Python)*.

## ***4.2 Descrierea componentelor***

Una dintre cele mai importante componente ale aplicației o reprezintă fișierele de la client, de care am amintit de nenumărate ori pe parcursul acestei lucrări.

Partea grafică este “prezentată” în cadrul acestor fișiere. Descrierea părții grafice este făcută prin intermediul codului de *Xml* prezent în aceste fișiere. Elementele generale care sunt prezente într-un astfel de fișier au fost prezentate în secțiunea “Fișiere” din cadrul capitolului de “Specificații”.

Modelul general de cod prezentat în cadrul capitolului de “Specificații” este o dovadă a faptului că aceste fișiere conțin cod de *Xml,* iar pentru a “transforma” aceste fișiere care în momentul de față nu sunt vizibile, în niște fișiere care să poată ulterior fi vizualizate, trebuie să lucrez cu acest cod*.*Cum orice document *Xml* ar trebui să respecte anumite reguli*.* Una dintre cele mai importante este aceea că: fiecare document *Xml* trebuie să aibă cel puțin un nod rădăcină, acest nod trebuie să conțină celelalte noduri ale documentului, am verificat dacă și în cazul acestor fișiere este respectată această regulă. Nodul rădăcină care apare în majoritatea fișierelor cu format *Xml*este *Panel,* dar mai sunt şi excepții în care nodul rădăcină este denumit *Container* şi nu *Panel.*

Codul pentru realizarea aplicației l-am realizat în *Python.* Pentru acest limbaj, am folosit extensia de *PyQt4*, deoarece este o combinație de *Python* și *Qt*necesară dezvoltării aplicației mele.Despre *PyQt* poate fi spus faptul că este o legătură între *Python* și *Qt.*PyQt are anumite caracteristici, care m-au influențat în decizia de a folosi această extensie, unele dintre ele ar fi:

−**procesare *Xml*** − este una dintre cele mai importante caracteristici care m-au determinat să folosesc *PyQt*-ul în cadrul aplicației mele.

Consider că nu aș fi putut folosi o extensie pentru Python mai bună decât PyQt care să poată procesa fișierele de *Xml* și care să aibă multe biblioteci de Qt care să poată fi folosite, așa cum are *PyQt*-ul. Biblioteca de Qt fiind importantă deoarece este una dintre cele mai puternice biblioteci *GUI*.

− **procesare de text, precum și afișarea de text −** această caracteristică reprezintă un avantaj, mai ales în cazul meu, deoarece doresc să afișez și conținutul fișierelor, deci foarte mult text.

Toate caracteristicile din cadrul *PyQt* sunt foarte importante, făcând din PyQt un *toolkit* ideal pentru crearea aplicațiilor de tip *GUI.*Un lucru important, care m-a ajutat foarte mult în crearea aplicației este acela că PyQt-ul conține foarte multe clase, funcții pe care le-am folosit în cadrul aplicației mele.

Aplicația creată de mine cuprinde 7 mari module, pe care le voi explica. Un lucru important, bine de știut, dar și de pus în practică este acela că o aplicație este bine să fie împărțită în module. În momentul în care o aplicație este împărțită în module, aceasta devine mai ușor de coordonat, iar în cazul în care se fac modificări în unul dintre module, aceste modificări nu vor afecta restul modulelor, desigur și dacă apare vreo problemă în cod, această structurare permite rezolvarea problemei într-un mod mai rapid și eficient.

Cele 7 module ale aplicației sunt:

• Fișiere PXA

• Dicționare PXA

• Reguli pentru crearea dicționarelor

• XLS

•Dicționare QML

• QML String

• USERCMD

Orice aplicație de *PyQt* presupune crearea unui obiect care să reprezinte obiectul aplicației, la fel am făcut și eu, am creat un astfel de obiect, iar pentru asta am importat anumite clase. Pentru afișarea pe *desktop* a unei ferestre m-am folosit de un proces existent deja în biblioteca, acesta este *sys*, fiecare aplicație folosind acest proces pentru generarea a unei ferestre în care ulterior va apărea o anumită interfață.

Pentru crearea obiectului aplicației am importat modulul de *GUI* denumit *QtGui.* Acest modul conține anumite componente și clase pentru partea grafică, denumite componente grafice, este folosit pentru crearea de butoane, ferestre și multe alte componente grafice. Aplicația obiect conține o serie de parametrii, specificați prin *sys.argv*.

Pentru crearea ferestrei care să fie apoi afișată pe *desktop* și să conțină numele aplicației am creat obiectul aplicației, un obiect pentru vizualizarea ferestrei propriu zise;am creat această fereastră în așa fel încât ea să se poată mări sau minimiza, deci este posibilă o redimensionare a ei, afișarea titlului, precum și închiderea acesteia atunci când se dorește.

### ***4.2.1 Fișierele PXA***

Fișierele pe care le-am primit, fiind în format *Xml*și având o extensie necunoscută, din cauza căreia fișierele nu puteau fi deschise, le-am convertit în fișiere cu extensia *Qml*, pentru aceasta, primul pas pe care l-am făcut în cadrul acestei aplicații a fost acela de a citi fișierele de *Xml* pentru a putea mai apoi să le prelucrez.

Pe lângă librăriile şi pachetele pe care le include acest limbaj de programare, se regăsesc şi cele impuse de sistem dar şi legate de *Xml*. Pentru a putea parcurge fișierele de *Xml* şi pentru a le prelucra (transformaîn dicționare), am folosit un pachet care aparține *Python*- ului şi anume:*Lxml*.

*Lxml*-ul este un pachet care conţine biblioteci de *C*, acestea fiind:*libxml2* şi *xslt*. Este o combinațiede viteză şi simplitate dintre *Xml* şi *API*-urile de *Python.* Un *API*(Application Programming Interfaces) este un set de de instrucțiuni şi standarde folosite pentru a accesa aplicații internet. *Lxml*-ul este compatibil cu *ElementTree API.*

În momentul instalării Python-ului, acest pachet nu este inclus, trebuind a fi instalat separat. Acesta poate fi descărcat din cadrul [***Python Package Index***](https://pypi.python.org/pypi/lxml).

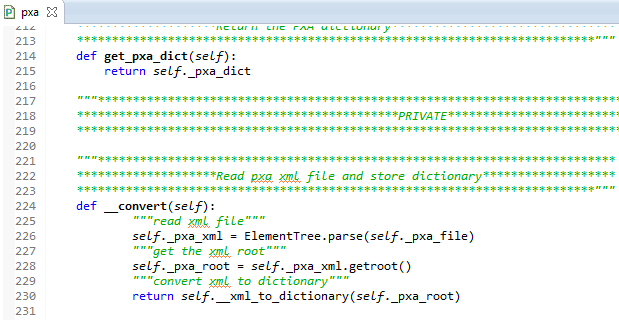
În cadrul aplicației, pentru a avea acces la bibliotecile existente în *Lxml*, am importat *ElementTree*, acesta fiind un obiect specific pachetului *Lxml* în cadrul căruia sunt stocate datele sub formă ierarhică. Am ales să mă folosesc de acest obiect, deoarece cu ajutorul lui pot parcurge fișierele în funcție de noduri. Este ideal pentru transformarea unui fișier *Xml*, fiecare element având un anumit număr de proprietăți asociate:

* Unui text de tip string.
* Unui tag care este un *string* fiind identificat printr-un element reprezentativ (tip element).
* Numărul de atribute stocate într-un dicționar de *Python.*
* Numărul de elemente copil stocate într-o secvență de *Python.*

Fiind un format de tip ierarhic, iar cea mai simplă modalitate de reprezentare este aceea sub formă de arbore în cadrul căruia există două clase:*ElementTree*reprezintădocumentul *Xml* sub forma unui arbore iar clasa *Element* reprezintă un singur nod dintr-un arbore.

Am realizat așadar o funcție de citire a fișierelor date în format *Xml,* în cadrul acestei funcții, scriind cod în *Python*,am putut să parcurg şi să iau fiecare nod în parte din cadrul codului de *Xml* şi cu ajutorul unor funcții predefinite din *Python* am reușit să extrag nodul rădăcină util pentru a ști cum o să se facă parcurgerea, după care, pentru a avansa şi pentru a ajunge de fapt acolo unde mi-am propus, am transformat fișierele cu extensia *pxa* şi format *Xml* în dicționare.

Funcția în care se realizează citirea acestor fișiere este expusă în Fig 4.2.1.1, numele ei fiind \_\_*convert*(). În cadrul ei, pentru citirea propriu zisă fișierul care doresc sa-l citesc este preluat, urmând să fie analizat conform *ElementTree.* Tot în cadrul acestei funcții am extras nodul rădăcină al fișierului, după efectuarea acestor operații, am putut să transform aceste fișiere *Pxa* în dicționare cu același nume.



*Fig 4.2.1.1 Citire fișiere Xml*

### ***4.2.2 Dicționare Pxa***

În Python există posibilitatea de a crea și de a accesa dicționare deja create, acest lucru fiind încă un avantaj al folosirii script-ului de *Python*. Fără existența dicționarelor din *Python* lucrurile nu ar fi decurs așa cum aș fi dorit, pentru că, valorile și cheile din aceste fișiere sunt dezordonate, ar fi fost imposibilă compararea elementelor existente în cadrul fișierelor *pxa* cu elemente specifice limbajului *qml,* iar transformarea acestor fișiere în fișiere *qml* ar fi fost imposibilă, nu puteam să folosesc *Qt-ul,* rezultatul fiind probabil foarte greu de obținut sau chiar imposibil.

Entitățile existente în cadrul fișierului sunt analizate în cadrul operației de citire, urmând transformarea acestora în dicționare prin realizarea unei operații de conversie, operație care nu modifică structura inițială a fișierelor.

Transformarea fișierelor în dicționare este o operație destul de complexă care a necesitat: studierea îndeaproape a fișierelor, “împărțirea” problemei în probleme mai mici. Pentru realizarea transformării propriu zise, am efectuat următorii pași:

− Am folosit o funcție de ștergere a *namespace*-urilor din fișierelor

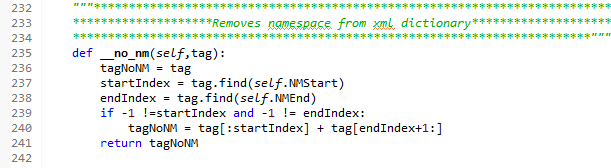
După această operație, în cadrul afișării, nu este nevoie ca *namespace*-urile prezente în fișiere să fie afișate, pentru că: fișierele trebuie să conțină doar elemente care trebuie convertite apoi în format *Qml* folosind niște reguli de conversie prin care se specifică echivalentul elementelor din fișierele *Pxa* în fișierele *Qml.*

*Namespace*-urile se folosesc de obicei pentru a nu se putea confunda două elemente care au același nume. Acesta fiind definit de ***xmlns***.

Un spațiu de nume (*namespace*) este un URI (*Uniform Resource Locator*) unic. Se presupune că orice poate avea acces la numele de domeniu.

În cazul fișierelor *Pxanamespace*-ul este un URI prin care se indică adresa clientului care furnizează specificațiile referitoare la *HMI* dar și specificații la nivel funcțional ale tuturor elementelor HMI.

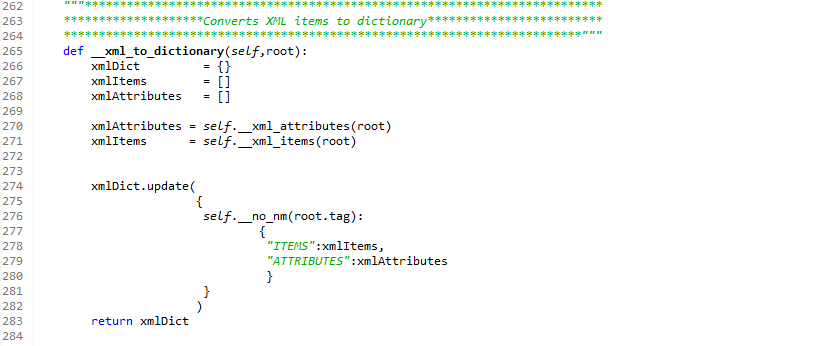
Funcția pe care am realizat-o pentru a șterge *namespace*-urile atunci când se dorește doar afișarea propriu zisă a elementelor este prezentată mai jos, în Fig 4.2.2.1.



*Fig 4.2.2.1 Ștergere namespace*

Pentru a șterge *namespace*-urile am folosit o funcție de căutare a *tag*-ului de început, precum și a celui de final, iar după condiția odată cu îndeplinirea condiției puse, au șters *namespace*-urile.

Următorul pas a fost acela de a prelua din fișier elementele de tip *item* și *atribute*și de a le introduce în dicționarul de *Xml*, a urmat convertirea elementelor din cadrul fișierelor *Xml*în fișiere *Xml.* Din Fig 4.2.2.2 se poate observa modul în care am realizat crearea dicționarului de Xml (adăugând *item*-urile și atributele extrase cu ajutorul funcțiilor menționate mai sus).Rezultatul obținut este un dicționar *Xml.*



*Fig 4.2.2.2 Dicționar Xml*

În cadrul funcției în care am realizat operația de citire a fișierelor am returnat și apelul la funcția care întoarce ca rezultat dicționarul de *Xml.* Fișierul de *Pxa*dorit, va fi transformat în dicționar de *Xml* care apoi va fi transformat în dicționar de *Pxa.* Pentru realizarea ultimei conversii am folosit o funcție în cadrul căreia am apelat funcția *\_\_convert()* valoarea returnată de aceasta fiind pusă în *\_pxa\_dict,* returnându-se în final dicționarul de *Pxa*.

### ***4.2.3 Reguli pentru crearea dicționarelor***

Pentru crearea unui dicționar din alt dicționar este nevoie de reguli, prin intermediul cărora să se specifice elementele din dicționarul de bază (de la care se pleacă) și echivalentul acestor elemente, pentru ca în momentul în care are loc conversia, un oarecare element să fie transformat corect în echivalentul său.

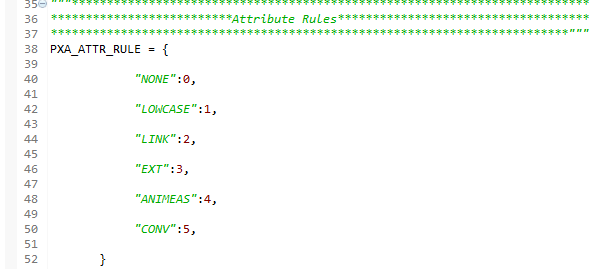
Am creat 3 seturi de reguli pe care le-am folosit ulterior la crearea dicționarului de *Qml.* Acestea sunt:

* Reguli generice

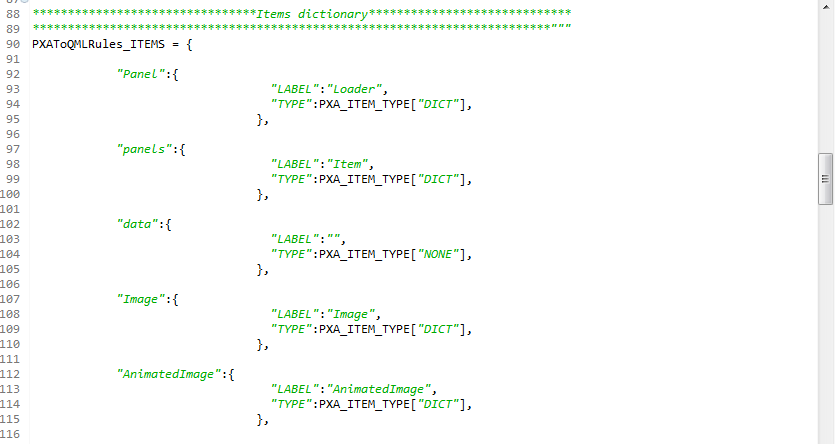
În cadrul acestui set apar: tipurile atributelor elementelor din dicționarul *Pxa*(pot fi de tipul: *int*, *bool*, *enum*, etc), reguli legate de aceste atribute (*none*, *link*, *lowcase*, etc), tipurile *item*-urilor existente în dicționare (*list*, *dict*, *attrib*), obiecte în *Qml* echivalente *item*-urilor din dicționarele de *Pxa*(*Image*, *ImageIndex*, *Container*, *Rectangle*), dicționarul de articole.

Acest “mic” dicționar conține o listă de elemente care apar în dicționarul de *Pxa*dar și echivalentul acestora în Qml − tipul atributelor specifice precum și felul acestora (dacă sunt de tip listă sau dicționar sau alt tip specificat în secțiunea de tipuri ale *item*-urilor).

Regulile generice au forma așa cum reies din Fig 4.2.3.1 și Fig 4.2.3.2



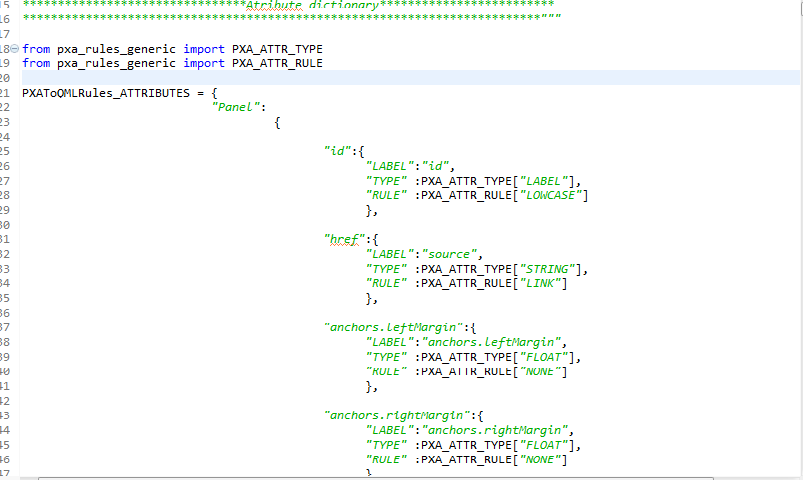
*Fig 4.2.3.1 Reguli ale atributelor*



*Fig 4.2.3.2 Dicționar Item*

Al doilea set de reguli conține reguli referitoare la atributele elementelor din dicționarele de *Pxa* și echivalentul acestora în dicționarele de *Qml*.

În cadrul acestui set sunt importate reguli definite în cadrul setului de reguli generice. Sunt importate și folosite reguli referitoare la tipul atributelor precum și reguli impuse acestor atribute. Așadar, pentru fiecare element din dicționarul de *Pxa* este specificat echivalentul său în *Qml,* fiind specificat pentru fiecare element în parte: atributul său precum și anumite proprietăți ale acestuia: tipul atributului, dar și alte reguli ale acestor tipuri. Acestea arată așa cum apare în Fig 4.2.3.3.



*Fig 4.2.3.3 Dicționar atribute*

Ultimul set conține reguli legate de enumerări referitoare la transformări, alinieri ale elementelor din dicționarul de *Pxa*, dar și echivalentul acestora în *Qml*. Exemplu de asemenea reguli este prezentat în Fig 4.2.3.4.



*Fig 4.2.3.4 Dicționar de enumerări*

### ***4.2.4 FișiereXLS***

Regulile definite le-am importat în fișiere *Excel* care le-am generat prin cod *Python* folosind biblioteci de *Python*.

Regulile au fost așadar împărțite în mai multe fișiere *Excel.*Fișierele de *Excel* le-am generat cu ajutorul unui generator de *Excel*. Pentru generarea acestor *excel*-uri, precum și a *template*-urilor care sunt de fapt fișiere de *Python* al căror “rezultat” este importat în fișierul în cadrul căruia are loc generarea de *excel*-uri, am folosit biblioteci specifice *Python*-ului.

Una dintre cele mai importante biblioteci care conține sintaxă de *template* este *Jinja2*. *Jinja2* este așadar un *designer* prietenos pentru lucrul cu *template*-uri, modelat după template-urile *Django*. Deoarece eu am folosit varianta de Python 2.7, *Jinja2*este compatibil cu aceasta varianta, astfel că *template*-urile sunt create corect. Pentru a putea folosi avantajele oferite de această bibliotecă, este necesară instalarea *Jinja2,* pentru asta am instalat un pachet care conține *Jinja2.*

În cadrul acestui modul am creat cu ajutorul bibliotecii *Jinja2* template-uri care conțin regulile definite în cadrul modului de reguli. Aceste *template*-uri sunt importate în cadrul fișierului în care are loc generarea de fișiere *excel*.

Generarea fișierelor am făcut-o cu ajutorul unui generator care creează linie cu linie un fișier de *excel*. Am creat pentru fiecare *spreadsheet* linii și coloane. Există *spreadsheet*-uri de mai multe tipuri, în funcție de regulile care trebuie să fie prezente în cadrul lui. Regulile preluate din *template*-uri nu sunt puse la întâmplare, fiecare regulă are un loc bine definit în cadrul unui *spreadsheet* dintr-un fișier de *excel*.

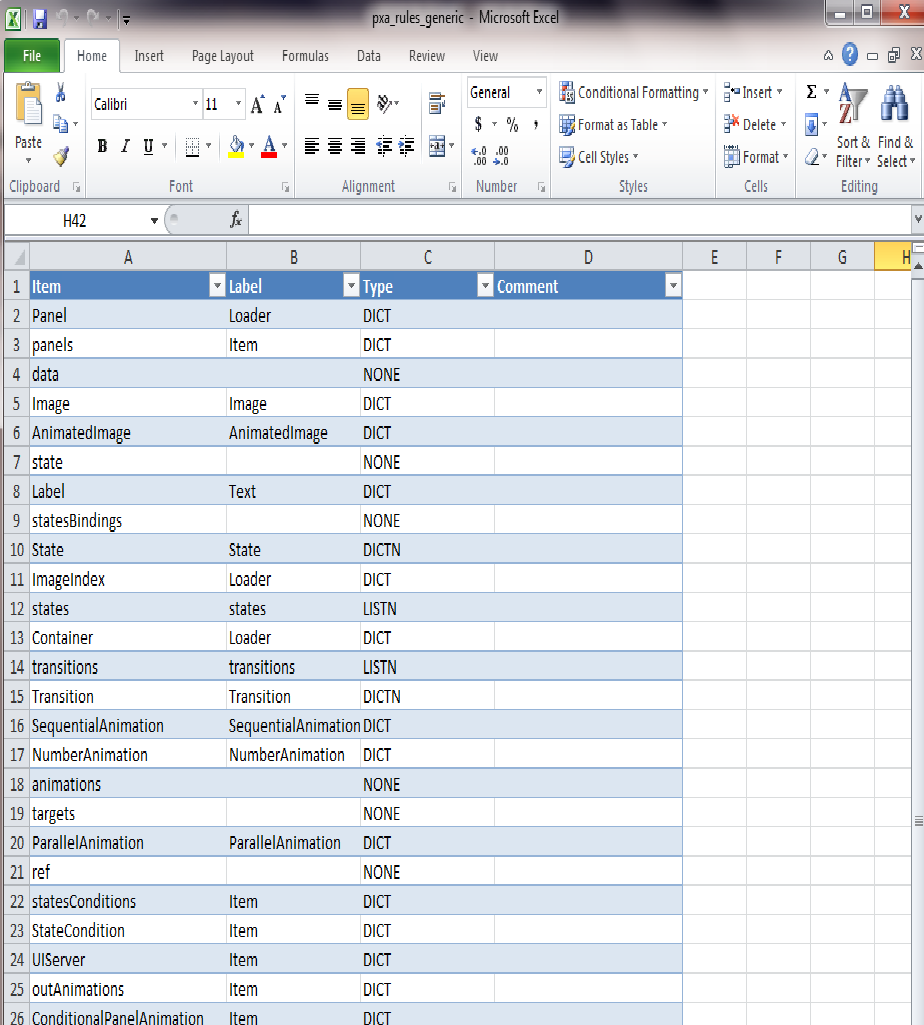
Am luat fiecare linie și fiecare coloană și am completat-o cu reguli specifice fiecărei coloane. Fiecare coloană are un nume, iar în cadrul său apar doar reguli specifice acelei coloane. Pentru fiecare element, atribut sau orice altă proprietate, am contorizat numărul regulilor care caracterizează aceste elemente, urmând ca și în cadrul *spreadsheet*-ului să fie completate un număr de celule egal cu numărul regulilor definite.

Fiecare linie și coloană are un index asemănător unei matrice, astfel că, în momentul în care am generat aceste fișiere și am transferat regulile în cadrul lor, am avut grijă ca în fiecare celulă să fie poziționat elementul potrivit.Fiecare linie “admite” introducerea atât a numerelor, cât și a unor șiruri de caractere de o anumită dimensiune. Numărul de fișiere de *excel* pe care le-am generat este egal cu numărul *template*-urilor generate din seturile de reguli.

Am generat astfel fișierul *excel* cu numele *pxa\_rules\_atribute.xls* care conține un număr de patru *spreadsheet*-uri, referitoare la regulile legate de imagini, de tranziția acestora, de proprietăți ale acestora, și de anumite animații. În cadrul fiecărui *spreadheet* apare un număr de 4 coloane, referitoare la numele atributului, tipului lui, precum și regulile de bază ale respectivului atribut.

Al doilea fișier numit *pxa\_rules\_enumeration* are un singur *spreadsheet* în cadrul căruia am afișat regulile specifice obiectelor de tip enumerare. Numărul de coloane este egal cu trei, în cadrul primei coloane apărând valorile din *Pxa* ale obiectelor de tip enumerare, în cadrul celei de-a doua apărând valorile corespunzătoare ale obiectului de tip enumerare în *Qml* (echivalentul din *Pxa*), iar ultima coloană este destinată comentariilor legate de aceste proprietăți și valori.

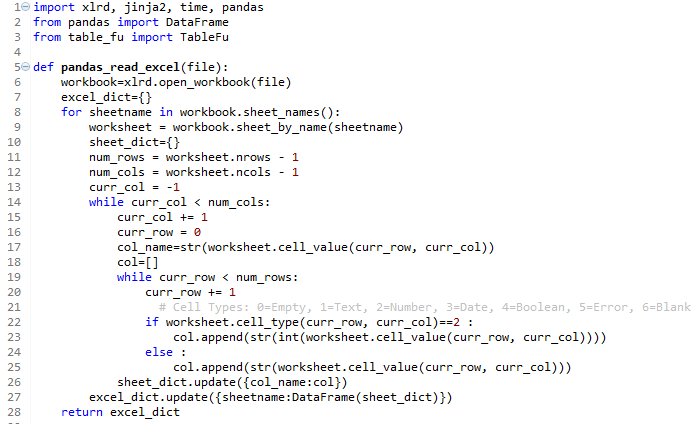
Ultimul fișier de *excel* denumit *pxa\_rules\_generic*conține un număr de cinci *spreadsheet*-uri referitoare la regulile generale: la tipulatributelor, regulile acestortipuri de obiecte prezente în cadrul unui fișier de *Pxa*, precum și echivalentul acestora în *Qml*. Fiecare *spreadsheet* are anumite coloane, în funcție de obiectul pentru care sunt afișate regulile.Fișierele de excel generate arată așa cum sunt prezentate în Fig 4.2.4.1.



*Fig 4.2.4.1 Fișierul Excel de reguli generice*

După ce am generat aceste fișiere de *excel*, le-am citit și le-am transformat în dicționare de Python, pentru a genera din acestea dicționarele de *Qml*. Pentru citirea acestor fișiere am folosit funcții din cadrul bibliotecii de Python numite *pandas*, care este deosebit de utilă în lucrul cu date structurate sub formă tabulară sau multidimensională. Cu ajutorul unor funcții din cadrul acestei biblioteci am reușit să citesc să ordonez datele din cadrul fișierelor și să le transform în dicționare de Python.

Funcția de citire a acestor fișiere de excel este prezentată în Fig 4.2.4.2.



*Fig 4.2.4.2 Citire fișier Excel*

### ***4.2.5 Dicționare Qml***

Acest modul are o importanță deosebită, deoarece în cadrul acestuia am transformat dicționarele de *Pxa* create din fișierele de *Pxa* în dicționare de *Qml*, cu ajutorul regulilor specificate.

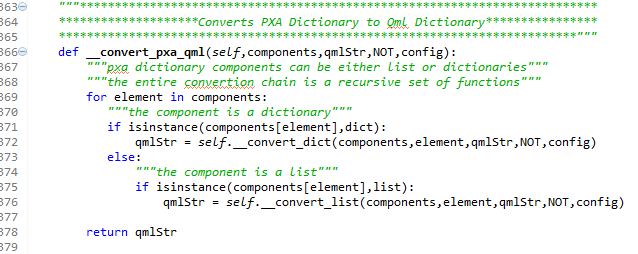
Pentru a transformarea dicționarelor de *Pxa* în dicționare de Python, am parcurs niște pași; aceștia sunt:

* Parcurgerea dicționarelor de *Pxa*
* Convertirea lor în dicționare de *Qml*

În funcție de rezultate obținute în urma parcurgerii acestor dicționare, am efectuat anumite operații pentru a transforma elemente din cadrul dicționarelor de Pxa, în elemente de Qml, folosindu-mă de seturi de reguli create în acest scop.

Dacă în urma parcurgerii am descoperit cazuri speciale cărora nu li se puteau aplica reguli, am luat fiecare caz în parte și l-am tratat. Din cauza cazurilor speciale menționate, am făcut o serie de funcții pentru fiecare caz în parte, pentru ca fiecare element “special” să fie convertitîn element de *Qml*, rezultând astfel dicționarul propriu zis de *Qml*.

Funcția propriu zisă de convertire a dicționarelor de *Pxa* în dicționare de *Qml* este prezentată în Fig 4.2.4.1.

**

*Fig 4.2.3.1 Convertire dicționare de Pxa în dicționare de Qml*

### ***4.2.6 Qml String***

Pentru a putea vizualiza conținutul fișierelor *qml* este necesară transformarea dicționarelor de *qml* în fișiere care să conțină string de *Qml* pentru a putea vizualiza în final rezultatul într-o formă clară și simplă, înțeleasă de orice utilizator.

Un prim pas pe care l-am făcut în operația de convertire a dicționarelor de *qml* în *string* de *qml* a fost acela de a parcurge aceste dicționare de *qml* obținute în urma conversiei dicționarelor de *pxa*.

Parcurgerea în cadrul acestor dicționare se face nod cu nod, iar pentru conversia propriu zisă nu doar am parcurs dicționarele, dar am luat fiecare element în parte și i-am “administrat” anumite condiții. În funcție de valoarea de adevăr ale acestor condiții *string*-ul obținut va avea o altă formă.

Luarea fiecărui element în parte din cadrul dicționarului de *Pxa* și transformarea lui în string de *Qml*, se realizează în cadrul mai multor funcții. În fiecare funcție realizată se transformă un element sau tip sau atribut în element *Qml*care să apară ca și *string*.

Funcțiile sunt sunt construite așa cum reies din Fig 4.2.6.1.



*Fig 4.2.6.1 Funcții specifice Qml String*

În cadrulacestor funcții am parcurs fiecare lista din dicționarul de pxa, am verificat indexul și am apelat funcția descrisă în cadrul modului de *Dicționare Qml*.

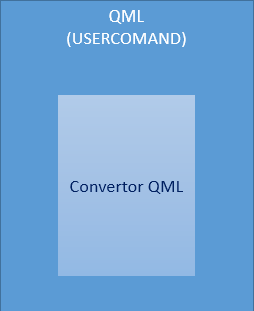
Am mai folosit funcții cu ajutorul cărora se pot introduce spații și comentarii din cod în cadrul fișierului de Qml.

### ***4.2.7 USERCOMAND***

Chiar dacă am reușit să transform fișierele cu extensia *pxa* în fișiere cu extensia *qml*, aceasta nu a fost suficient, a fost nevoie de un modul prin care utilizatorul să poată controla procesul, a fost nevoie de o interfață simplă care să ajute utilizatorul.

Am creat așadar o interfață ușor de folosit, manualul de utilizare fiind prezentat în cadrul capitolului de *Utilizare*.

Acest modul l-am gândit ca fiind ultimul din cadrul aplicației, dar și cel care înglobează tot ce am creat până acum. Schema care stă la baza acestui modul este prezentată în Fig 4.2.7.1.



*Fig 4.2.7.1 USERCOMAND*

După construirea schemei acestui modul și după construirea logicii, am implementat în Python codul corespunzător, cu ajutorul căruia acest modul va prinde contur.

Am creat o clasă separată în Python care conține date și metode menite să transforme acest concept într-o interfață cu care interacționează utilizatorul. Prima funcție este aceea de a crea o fereastră în cadrul căreia va apărea rezultatul fișierelor de *pxa* convertit în *qml.*

Funcția cu ajutorul căreia se va crea fereastra este prezentată în Fig 4.2.7.2.

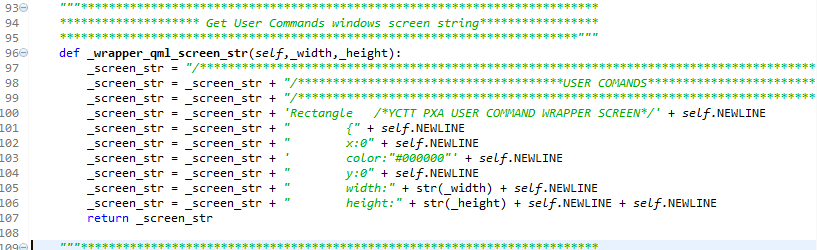
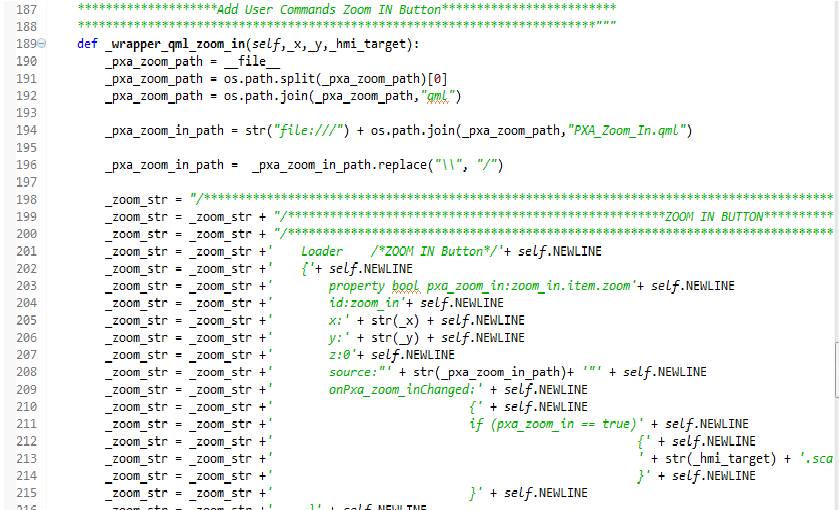
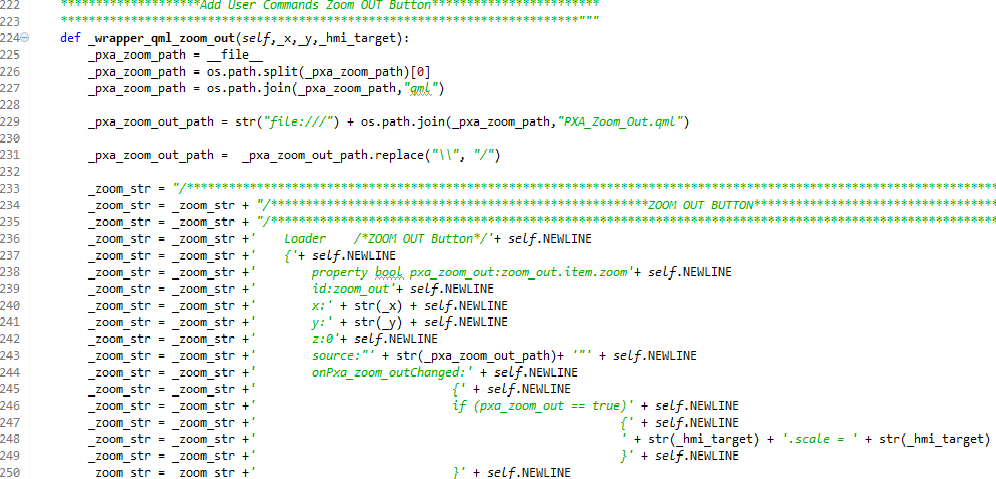


Fig 4.2.7.2 Fereastra USERCOMAND

În cadrul clasei am mai folosit funcții care preiau fișierul convertit și îl afișează sub forma imaginii corespunzătoare, care poate acum fi vizualizată de către utilizator. Pentru realizarea butoanelor am creat următoare funcții prezentare în Fig 4.2.7.3 și Fig 4.2.7.4.



*Fig 4.2.7.3 Buton zoom in*



*Fig 4.2.7.4 Buton zoom out*

Toate funcțiile create atât în cadrul acestei acestei clase, cât și in cadrul clasei generale (care înglobează alte clase și funcții) sunt foarte importante, dacă nu le-aș fi implementat pe toate, nu aș fi putut finaliza aplicația.

# **5. Utilizarea sistemului**

Orice program, fie el *software*sau de altă natură, trebuie sa aibă un manual de utilizare care să fie ca un ghid pentru cel ce-l răsfoiește și urmează să folosească produsul respectiv. În acest manual trebuie să fie precizate instrucțiuni referitoare la punerea în funcțiune și folosirea programului sau aparatului.

Dacă ar fi să fac un scurt ghid utilizare al aplicației mele (al convertorului utilizat de mine) pe care utilizatorul ar fi recomandat să-l parcurgă înainte de folosire, acesta ar conține următoarele capitole:

## ***5.1 Scurtă prezentare***

Prezentul Ghid de Utilizare se referă la aplicația “Convertor pentru limbajul descriptiv HMI”, și se recomandă parcurgerea lui înainte de utilizarea aplicației.

Aplicația este folosită pentru a vizualiza fișierele care inițial aveau extensia .*pxa*și care nu puteau fi vizualizate, doar pentru că nu exista o soluție pentru deschiderea lor. Datorită acestei aplicații, fișierele vor putea fi vizualizate, iar cei nevoiți să folosească conținutul acestor fișiere o vor putea face, putând totodată să-și continue și treaba.

Aceasta aplicație poate fi așadar folosită gratuit doar în cadrul companiei și este disponibilăoricărei persoane din cadrul firmei care are nevoie de aceasta.

Pentru ca aplicația să poată fi utilizată este necesară instalarea pe calculatorul fiecărui utilizator a unor programe.

## ***5.2 Instalare***

Fiecare utilizator va trebuie să instaleze următoarele programe:

- mediul de dezvoltare Eclipse *Kepler* împreună cu *plugin*-ul *PyDev* pentru a putea vedea codul scris și pentru a adăuga în script-ul de *Python* calea spre fișierul pe care dorește să-l convertească, dar și pentru a rula aplicația.

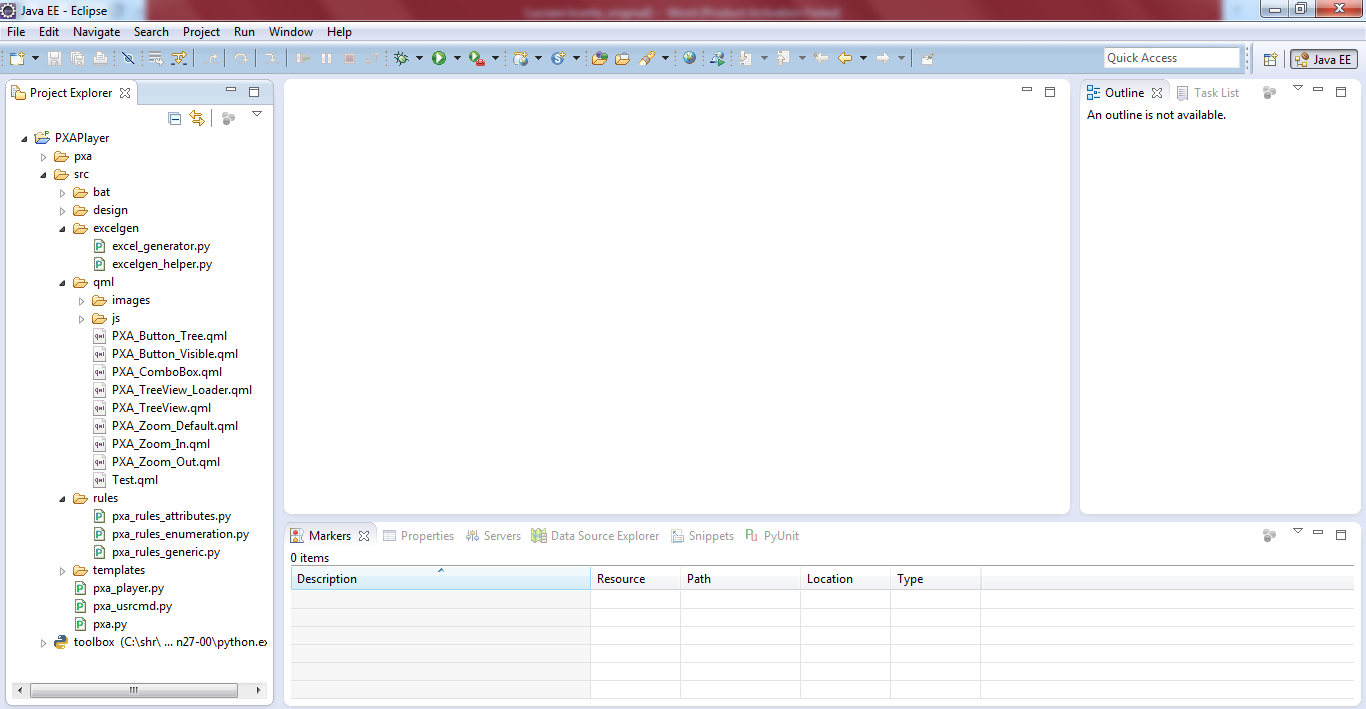
-mediul Qt 5.1 pentru a putea vizualiza fișierele convertite.

## ***5.3 Utilizare***

Odată instalate programele de care este atâta nevoie, este timpul ca aplicația să fie utilizată în scopul pentru care a fost creată. Așadar fiecare utilizator după ce a instalat aceste programe și după ce a copiat pe calculatorul personal *folder*-ul care conține aplicația trebuie să efectueze următorii pași:

− Să deschidă mediul de dezvoltare Eclipse și din meniu să aleagă File->Import, prin selectarea acestei opțiuni proiectul realizat va fi adus din locul în care este salvat în acest mediu pentru a putea fi vizualizat conținutul fișierului care s-a creat în momentul în care s-a creat aplicația. Utilizatorul trebuie să fie atent în momentul în care trebuie să introducă calea spre proiectul salvat. Așadar, după introducerea corectă a căii, trebuie să selecteze proiectul, acest lucru este necesar, deoarece există probabilitatea ca să mai existe mai multe proiecte care pot fi aduse din locul unde este și proiectul necesar a fi importat. După aceasta urmează ca utilizatorul să dea *click* pe butonul *Next,* pentru a putea fi important proiectul deja existent.

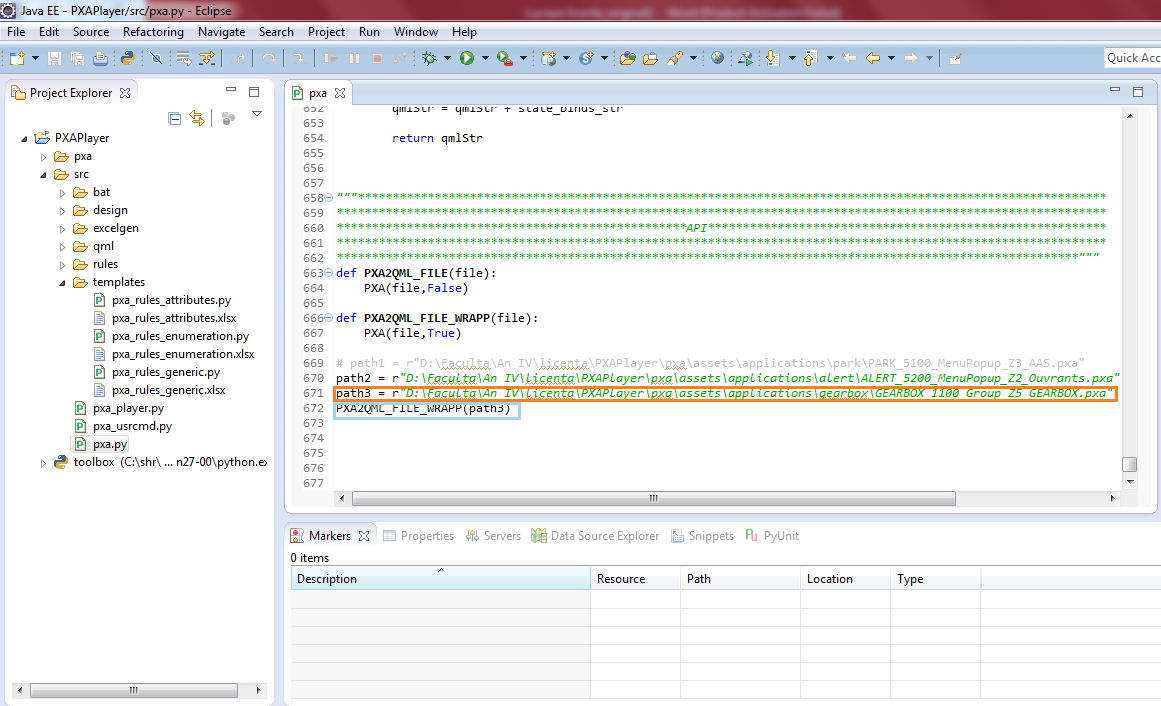
În momentul acesta, proiectul este adus din locul selectat, iar pe *desktop* ar trebui să vadă următoarea fereastră:



*Fig 5.3.1 Mediul de dezvoltare Eclipse*

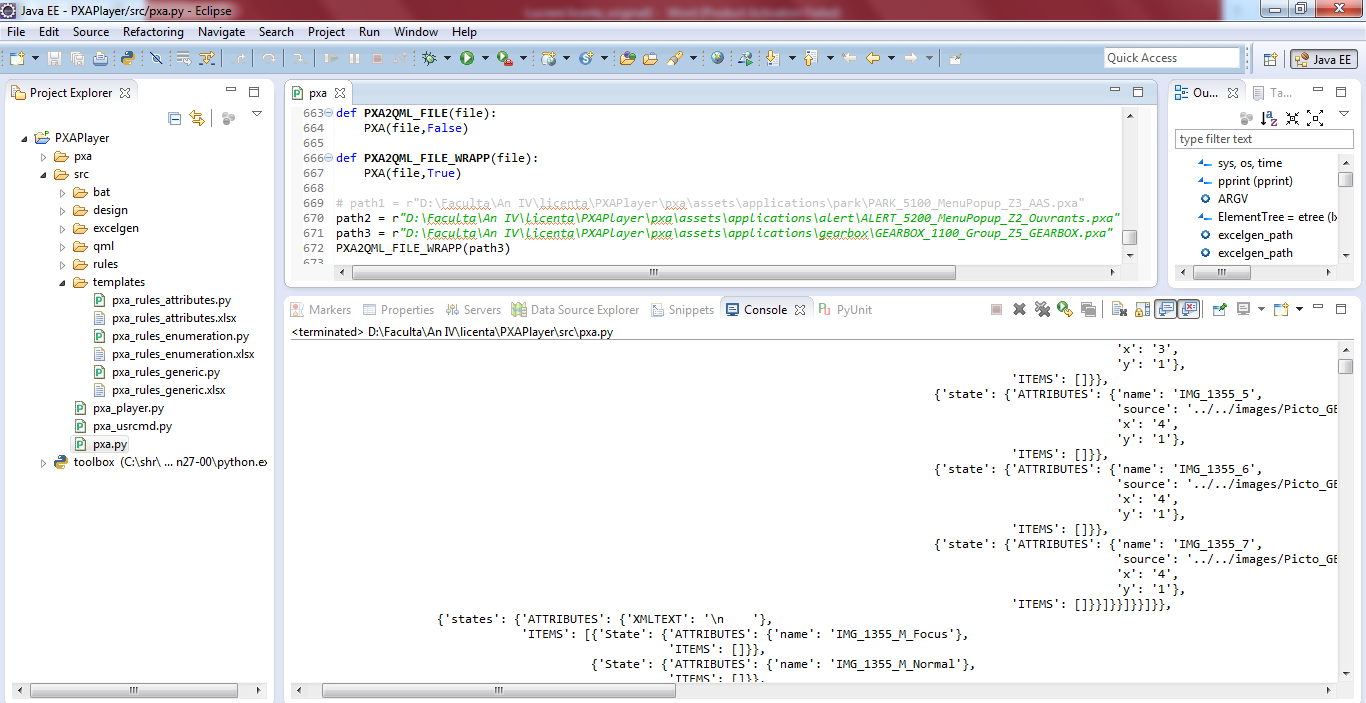
Se poate observa faptul că proiectul importat conține mai multe fișiere. Utilizatorul nu are dreptul de a modifica aceste fișiere care conțin cod de *Python*. Trebuie să deschidă fișierul cu denumirea *src*, care conține mai multe fișiere dintre care unul cu numele *templates* (acesta trebuie deschis). În momentul în care acest fișier este deschis se poate observa că el conține mai multe *script*-uri de *Python,* dintre acestea trebuie deschis *script*-ul cu numele *pxa.py.* În cadrul acestuia, singurul lucru care trebuie să-l facă este acela de a modifica calea spre fișierul care dorește să fie convertit pentru a putea fi vizualizat în final.

În codul din acest *script* o să trebuiască adăugată calea spre fișierul care urmează să fie convertit. În Fig 5.3.2 este specificat printr-un chenar roșu exact linia de cod care trebuie modificată, desigur dacă utilizatorul va dori să poată converti mai târziu și alte fișiere, poate să introducă și alte căi, însă în linia pentru convertire trebuie introdusă numele unei singure variabile în care s-a salvat calea spre un anumit fișier. Iar în momentul rulării va avea loc conversia propriu zisă, iar rezultatul va fi afișat așa cum apare în Fig 5.3.3



*Fig 5.3.2 Introducerea căii fișierului*

Mai jos va fi expus rezultatul obținut în momentul în care are loc conversia.



*Fig 5.3.3 Rezultatul conversiei*

Acest rezultat nu ar trebui să sperie utilizatorul, aceasta este dovada că fișierul pe care el dorea să-l convertească a fost convertit, acesta fiind codul de *qml*, noul cod care va apărea dacă fișierele vor fi deschise doar pentru a vizualiza codul din interior. Codul este de *qml* pentru că fișierele convertite vor avea extensia *qml.* Utilizatorul nu trebuie să înțeleagă acest cod, trebuie doar să fie convins că fișierul pe care va dori să-l vadă a fost convertit și în scurt timp va putea chiar să vizualizeze conținutul acelui fișier.

După conversia fișierului, utilizatorul trebuie să mai facă un ultim pas și anume:

− Trebuie deschis *Qml Viewer* care se poate găsi într-un singur mod și anume:

− Se urmează calea: Qt->5.1.1->mingw48\_32->bin->qmlviewer.

− Se deschide fereastra de Qml Viewer care trebuie să arate ca în Fig 5.3.4.

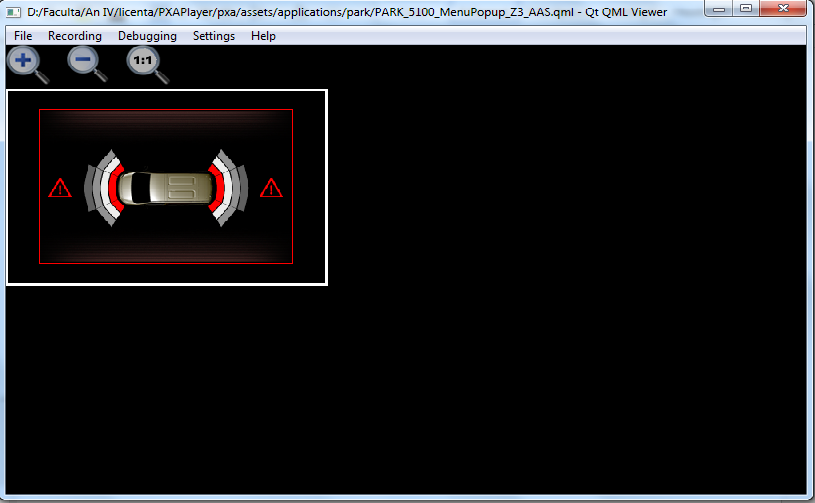


*Fig 5.3.4 Fereastra Qml Viewer*

Pentru a putea vizualiza imaginile din cadrul fișierului tocmai convertit, este necesar ca utilizatorul să acceseze *File*->*Open*și să adauge calea spre fișierul convertit. Fișierele convertite se află în: fișierul *PXAPlayer*, mai exact în locația: *PXAPlayer->pxa->assets->applications*, selectându-se și unul dintr modulele existente, în cadrul cărora se află fișierele convertite.

În urma acestei acțiuni o să apară o interfață foarte simplă cu utilizatorul în cadrul căreia o să apară conținutul fișierului (imaginea mult așteptată).

Rezultatul pe care îl va vedea în final utilizatorul, va fi cel așteptat, iar în Fig 5.3.5 va fi prezentat rezultatul unui fișier *pxa*, transformat în *qml*.



*Fig 5.3.5 Interfața cu utilizatorul*

Utilizatorul poate mări sau micșora imaginea, după cum dorește și poate reveni oricând la varianta inițială a imaginii. Un exemplu de mărire a imaginii este prezentat în Fig 5.3.6.

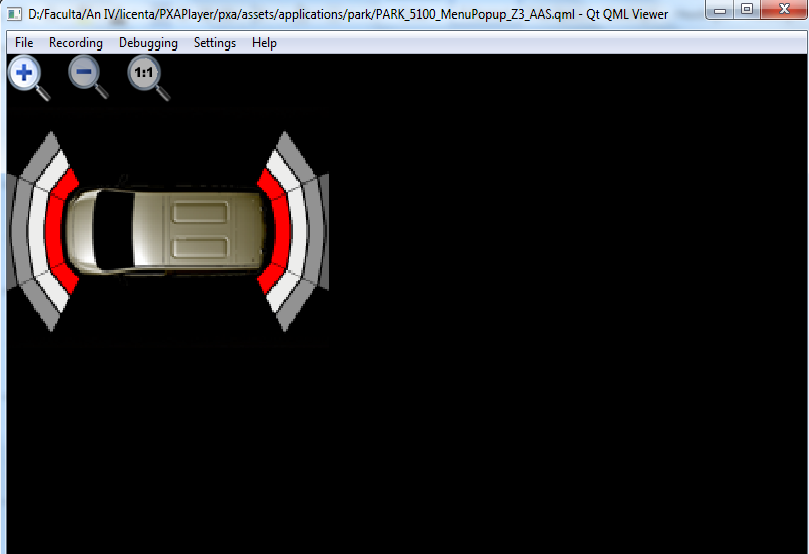


Fig 5.3.6 Mărirea imaginii

# ***6. Realizarea, punerea în funcțiune și rezultate experimentale***

Momentul în care are loc punerea în funcțiune precum și aflarea rezultatelor experimentale, reprezintă un moment important dar și tensionat în orice domeniu. În acest capitol o să prezint punerea în funcțiune a aplicației mele pentru a urmări și testa funcționalitatea sistemului prin date experimentale.

Voi lua un exemplu concret și anume:în momentul în care se dorește să se afișeze pe *display*o anumită imagine sau un grup de imagini, sunt foarte importante specificațiile, din acest motiv fără specificațiile grafice nu se poate face nimic, chiar dacă sunt oferite toate detaliile și condițiile care trebuie să fie îndeplinite în momentul în care trebuie afișat pe display, fără imaginile care trebuie să fie afișate nu se poate face nimic. Așa cum am mai spus imaginile trebuie prelucrate și trebuie făcute niște legături pentru ca ele să apară pe *display*.

Fișierele primite de la client, pe care le-am convertit sunt împărțite în mai multe module. În mod normal, în bordul unei mașini, sunt afișate multe imagini, grupuri de imagini, poate chiar text sau filmulețe, depinde de mașină. Noi însă trebuie să respectăm întru-totul cerințele impuse de client.

Aceste module sunt: alert, drive, gauge, gearbox, generic, hmi, navigation, obc, onoff, park, settings.

Fiecare modul în parte are o anumită specificație, dar indiferent de specificația acestuia, trebuie testată funcționalitatea pentru fiecare modul în parte.

Daca se dorește ca pe *display* să fie afișată o poză din modulul*alert* prin care să se semnalizeze faptul că o ușă este deschisă, trebuie făcute o grămadă de lucruri și trebuie scris foarte mult cod ca să se ajungă aici. Unul dintre primii pași care trebuie făcuți este acela de a vedea ce poză din cadrul acestui modul este cea potrivită și care îndeplinește anumite condiții. Pentru asta, în primă fază se convertesc fișierele din cadrul modului *alert*, după care, ar trebui să se verifice fiecare fișier convertit pentru a se vedea ce poză conține. Trebuie verificat dacă poza este cea potrivită însă fișierele au nume sugestive, din această cauză se poate observa cu ușurință care ar trebui să fie fișierul care ar trebui să conțină o poză.

Pentru a vizualiza fișierul deschidem *Qml Viewer,* și cu ajutorul File->Open deschidem fișierul cu numele sugestiv *ALERT\_5200\_MenuPopup\_Z2\_Ouvrants* și vom observa faptul ca apare poza din Fig 6.1.

Se mai poate observa și faptul că poza poate fi mărită, micșorată sau se poate reveni la starea inițială în cazul în care se dorește acest lucru. În Fig 6.2, Fig 6.3 se observă faptul că sunt posibile și aceste opțiuni menționate.

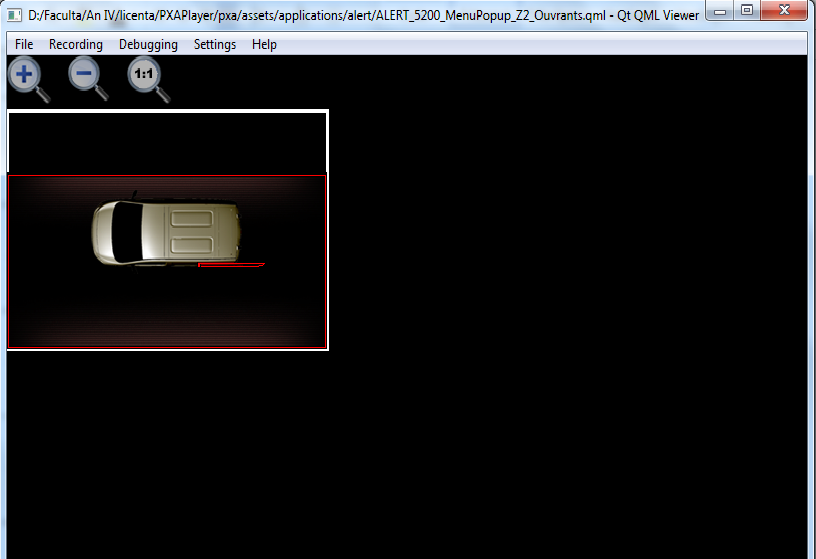


Fig 6.1 Fișierul *ALERT\_5200\_MenuPopup\_Z2\_Ouvrants*



Fig 6.2 Fișierul *ALERT\_5200\_MenuPopup\_Z2\_Ouvrants*(*rezultat mărit)*



Fig 6.3 Fișierul *ALERT\_5200\_MenuPopup\_Z2\_Ouvrants*(rezultat micșorat)

# **7. Concluzii**

În această lucrare s-a urmărit implementarea unui convertor pentru limbajul descriptiv *HMI.* Acesta aplicație realizată de mine este o parte a unui proiect din cadrul companiei Yazaki Component Technology.

Am ales să implementez și desigur am implementat acest convertor, pentru că nu există unul, iar fișierele primite de la client nu puteau fi vizualizate, acest fapt reprezentând o piedică în calea inginerilor de HMI care au ca sarcină realizarea părții grafice. Sarcina acestora a fost ușurată pentru că de acum înainte, nu vor fi nevoiți să piardă timpul pentru a căuta soluții greu de găsit, vor putea vizualiza conținutul fișierelor (pozele din interior) și le vor putea modifica și prelucra în programul folosit pentru acest lucru, numit *Populus*. Imaginile prelucrate ajung ulterior pe *LCD*-ul din cadrul unui bord de mașină.

Modulele implementate sunt toate funcționale, aplicația întreagă este funcțională, aceasta fiind testată de inginerii de *HMI* de nenumărate ori, chiar din momentul finalizării aplicației. Nu au apărut probleme în timpul utilizării. Modul de utilizare nu a ridicat probleme nimănui, aceasta datorându-se interfeței care este una simplă și prietenoasă cu utilizatorul.

Deși am realizat această aplicației pentru un anumit proiect, în cadrul căruia a fost neapărată nevoie de ea, aceasta poate fi folosită și după terminarea acestuia, în cadrul altui proiect. Oricare ar fi limbajul descriptiv din viitoarele proiecte, bazat pe *Xml* și concepte asemănătoare *Qml*-ului convertorul poate fi folosit de fiecare dată prin simpla modificare a regulilor dicționarelor.

Unele dintre cele mai importante proprietăți ale unei aplicații software sunt: flexibilitatea, reutilizabilitatea și faptul dacă aceasta este generică sau nu. Aplicația realizată de mine este generică, flexibilă deoarece prin simple modificări făcute în cadrul acesteia devine funcțională pentru orice proiect și reutilizabilă deoarece se poate reutiliza de fiecare dată.

# **8.Bibliografie**

[1]<http://python.net/~gherman/propaganda/tut-ro/tut-ro.pdf>

[2]<http://www.w3schools.com/xml/xml_whatis.asp>

[3] <http://vechi.upg-ploiesti.ro/col/ldumitrascu/pdf/xml/XML.pdf>

[4]<http://www.slideshare.net/dragosacostachioaie/mediul-de-programare-multiplatforma-qt>

[5] <http://ro.wikipedia.org/wiki/Structur%C4%83_de_date>

[6] <http://qt-project.org/doc/>

[7]<http://qt-project.org/doc/qt-4.8/qdeclarativeintroduction.html>

[8] <http://zetcode.com/gui/pyqt4/introduction/>

[9] <https://en.wikipedia.org/wiki/Affective_computing>

[10] Affective Computing by [Rosalind W. Picard](http://www.goodreads.com/author/show/802927.Rosalind_W_Picard)

[11] <http://www.journals.elsevier.com/computer-vision-and-image-understanding/>