

Universitatea POLITEHNICA din Bucureşti
Facultatea de Automatică și Calculatoare,
Departamentul de Calculatoare



LUCRARE DE DIPLOMĂ

Managementul elementelor vizuale
pentru aplicații mobile

Conducător Științific:
Prof.dr.ing. Nicolae Țăpus

Autor:
Claudia Rafaela Rogoz

Bucureşti, 2016

Abstract

Lucrarea de față propune un model de management al elementelor vizuale între două aplicații destinate designerilor de user experience. Proiectul este realizat architectural sub forma unui plugin pentru aplicația Adobe Experience Design.

Prin intermediul acestui proiect, se dorește realizarea unei scheme de management atât pentru aplicații mobile, cât și pentru site-uri web. Astfel, un developer de iOS va putea crea, actualiza și modifica entitățile grafice - imagini, text, scene, tranzitii între ecrane - ale aplicației iOS dezvoltate.

Soluția propusă va realiza translatarea între două reprezentări specifice dezvoltării de aplicații iOS - pe de o parte XCode, iar pe de altă parte Adobe Experience Design.

Contents

Abstract	ii
1 Introducere	1
1.1 Descrierea proiectului	1
1.1.1 Scop și motivație	1
1.1.2 Obiectivele Proiectului	1
2 Prezentarea structurilor XCode și Adobe Experience Design	3
2.1 Prezentare XCode	3
2.1.1 Structura unui fișier .storyboard	4
2.2 Prezentare Adobe Experience Design	7
2.2.1 Structura unui fișier .xd	7
2.3 Translatarea și Scalarea dimensiunilor elementelor vizuale	9
3 Prezentarea schemelor de translatare	11
3.1 Translatarea din XCode în Adobe Experience Design	11
3.1.1 Translatarea Switch-ului și a altor elemente grafice similare	14
3.1.2 Translatarea Interacțiunilor XCode->XD	15
3.2 Translatarea din Adobe Experience Design în XCode	16
3.2.1 Translatarea Path-urilor și a diverselor figuri geometrice	18
3.2.2 Translatarea textului	21
3.2.3 Translatarea grupurilor	21
3.2.4 Translatarea imaginilor	22
3.2.5 Translatarea interacțiunilor	23
3.3 Sincronizarea scenelor între cele două reprezentări	23
3.3.1 Sincronizarea Adobe Experience Design - XCode	23
4 Implementare	25
4.1 Arhitectură	25
4.1.1 Demo	27
4.2 Performanță	28
4.2.1 Comparație Scheme de translatare	29
5 Concluzii	31
5.1 Îmbunătățiri ulterioare	31

Chapter 1

Introducere

Proiectul de față este un produs software care vine în ajutorul developerilor și designerilor de aplicații user experience. În acest sens, se dorește realizarea unui management al asseturilor dintre două produse software - IDE-ul XCode și Adobe Experience Design.

În capitolul acesta, se va realiza o scurtă descriere a proiectului, motivația și obiectivele aplicației, iar în urmatoarele secțiuni se va detalia arhitectura și modul de implementare al acesteia.

1.1 Descrierea proiectului

Lucrarea are ca temă realizarea managementului elementelor vizuale prin găsirea unei scheme de translatare între diferite reprezentări.

O primă reprezentare este una definită prin Xcode Interface Builder. Cea de-a doua reprezentare este definită prin produsul software Adobe Experience Design (XD). Pentru implementarea acestui proiect s-a luat în considerare structura internă a fișierelor corespunzătoare celor 2 aplicații.

1.1.1 Scop și motivație

Principalul scop al acestui proiect este construirea unei scheme de translatare între două produse software destinate designerului și prototipurilor de aplicații mobile și site-uri web, pentru a veni în ajutorul developerilor de iOS. Prin intermediul soluției descrie, crearea, sincronizarea, actualizarea, dar și colaborarea între membrii unei echipe de dezvoltare de aplicații se va realiza facil, asigurându-se o automatizare a modului de lucru.

Un scop secundar este găsirea unei soluții eficiente de reprezentare și de translatare între cele două formate interne XCode-ului, respectiv XD-ului, pentru a fi folosit și în alte domenii de aplicații.

Finalitatea acestei aplicații este crearea automată a unei “punți” între cele două tehnologii menționate mai sus, dedicate designerilor de UX.

1.1.2 Obiectivele Proiectului

Obiectivul principal al acestei teme este acela de translatare între un produs software dedicat dezvoltării unei aplicații (XCode) și o aplicație de design și de realizare de prototipuri (Adobe XD). Totodată, prin intermediul acestui proiect se dorește construirea automată a aplicației

XCode, pe baza prototipului obținut prin XD. Astfel, soluția propusă va trebui să asigure următoarele funcționalități:

1. Soluția va permite unui dezvoltator iOS să creeze un fișier de design Adobe Experience Design care să conțină elementele vizuale din proiectul său iOS (din XCode)
2. Un utilizator poate modifica ecranele aplicatiei, elementele vizuale și tranzițiile pentru fișierul Adobe Experience Design creat mai sus. De asemenea, se pot crea atât noi elemente vizuale, cât și tranziții.
3. Soluția va permite dezvoltatorului de iOS să-ți actualizeze proiectul său iOS pe baza ultimelor modificări aduse fișierului Adobe Experience Design creat. Sincronizarea se va putea face automat, în momentul în care se salvează fișierul de design.
4. Utilizatorul va putea să compileze aplicația sa iOS din XCode, iar modificările sunt văzute corespunzător.

De asemenea, un obiectiv secundar al acestui proiect este construirea unei scheme de translatăre generică între cele două reprezentări specifice XCode-ului și Adobe XD-ului: xml, respectiv json. Astfel, soluția propusă se poate extinde, putând fi folosită în diferite domenii de aplicații.

Chapter 2

Prezentarea structurilor XCode și Adobe Experience Design

Așa cum s-a precizat în capitolul anterior, proiectul reprezintă un model de translatare între două aplicații, care va fi construit sub forma un plug-in adus aplicației Adobe Experience Design, pentru interacțiunea cu IDE-ul XCode. Pentru a putea construi această translatare, se vor explica, în acest capitol, structurile și modul de lucru corespunzătoare celor două aplicații. În primul rând, însă, se vor defini noțiunile de elemente vizuale/grafice.

Elementele vizuale, atât în XCode, cât și în Adobe experience Design, sunt reprezentate prin mulțimea de asset-uri care se pot reprezenta în mod vizual pe suprafața de design (text, imagini, butoane, interacțiuni și scene). Fiecare element vizual are specificat un comportament specific, în funcție de categoria de asseturi în care se încadrează.

2.1 Prezentare XCode

XCode este un IDE realizat de Apple care conține o serie de tool-uri pentru dezvoltarea de software pentru OS X, iOS, WatchOS și tvOS. XCode suportă cod sursă pentru o serie de limbaje de programare (C, C++, Objective-C, Objective-C++, etc.), dar și pentru o serie de modele de programare (Cocoa, Carbon, Java, etc.). Aplicația principală a suitei este IDE-ul, numit de asemenea XCode. Suite XCode include pe lângă documentația Apple de dezvoltare și o aplicație built-in - Interface Builder, o componentă principală în dezvoltarea soluției propuse.

Interface Builder este o aplicație de development de software pentru sistemul Apple Mac OS X, care permite dezvoltatorilor de Cocoa și Carbon să creeze interfețe pentru aplicații folosind interfață grafică. Interfața rezultată va fi stocată într-un fisier .nib (NeXT Interface Builder), sau .xib.

Interface Builder oferă palete de culori, colecții de elemente vizuale - text, tabele, meniuri, butoane, etc - necesare dezvoltatorilor de aplicații ObjC. Astfel, prin intermediul editorului Interface Builder, crearea interfeței grafice pentru utilizator se realizează ușor, fără a fi nevoie de cod efectiv. La baza managementului de elemente vizuale se află Drag Drop spre canvas-ul de lucru.

Deoarece Cocoa și Cocoa Touch sunt construite folosind modelul Model-View-Controller, interfețele utilizator se pot realiza independent de implementările lor. Aceste interfețe vor fi stocate în fișierele .nib, iar la compilare, se vor crea dinamic legăturile între UI și implementare.

O aplicație iOS este compusă din multiple scene între care un utilizator navighează. Relațiile dintre aceste scene sunt definite de fișierele .storyboard, care vor reprezenta întregul flow al aplicației. Prin intermediul Interface Builder, storyboard-ul este actualizat conform elementelor grafice afișate, prin crearea, modificarea și realizarea traițiilor dintre scene.

XCode permite o serie de controlere pentru storyboard: Table View Controller, Collection View Controller, Navigation Controller, Tab Bar Controller, Page View Controller, GLKit View Controller sau Custom View Controller, însă pentru acest proiect soluția propusă rezolvă doar controllerul generic - View Controller. Pentru extinderea acestei soluții și pentru alte tipuri de controlere, va trebui extinsă schema de translatare care va fi prezentată ulterior, conform specificațiilor.

2.1.1 Structura unui fișier .storyboard

În această secțiune se va explica anatomia de bază al fișierului .storyboard XML și a câtorva tag-uri principale: *scene*, *viewController*, *view*, *segue*. De asemenea se vor explica modul de interacțiune între entități și modul în care acestea lucrează împreună pentru construirea interfețelor utilizator din Interface Builder. Un fișier “.storyboard” mapează elementele vizuale într-o reprezentare de tip xml. Așadar, fiecărui element vizual îi corespunde un tag cu atribute care specifică caracteristicile acestuia în mod formal. Întrucât tagurile au o reprezentare ierarhică, atributele acestora depind de tag-urile părinte. De exemplu, un tag de încadrare al unui element vizual este definit relativ la tagurile sale părinte de încadrare.

În momentul în care un nou storyboard este creat, se va construi un fișier xml cu tag-ul părinte *document* și cu o serie de atribute care conțin informație meta (figura 2.1):

```

1 <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2 <document type="com.apple.InterfaceBuilder3.CocoaTouch.Storyboard.
  XIB" version="3.0" toolsVersion="9532" systemVersion="15E65"
  targetRuntime="iOS.CocoaTouch" propertyAccessControl="none"
  useAutolayout="YES" useTraitCollections="YES">
3   <dependencies>
4     <deployment identifier="iOS"/>
5     <plugIn identifier="com.apple.InterfaceBuilder.
    IBCocoaTouchPlugin" version="9530"/>
6   </dependencies>
7   <scenes/>
8 </document>

```

Figure 2.1: Document tag

Atributul **targetRuntime** specifică sistemul de runtime în care storyboard-ul este folosit. *iOS.CocoaTouch* specifică faptul că storyboard-ul este de tipul iOS.

Tag-ul **dependencies** identifică orice dependență necesară storyboard-ului. Tag-ul copil **plugin** specifică pluginul necesar acestui storyboard.

Tag-ul **scenes** conține toate scenele (view controllers) din storyboard. Reprezentarea xml corespunzătoare se poate observa în figura 2.2

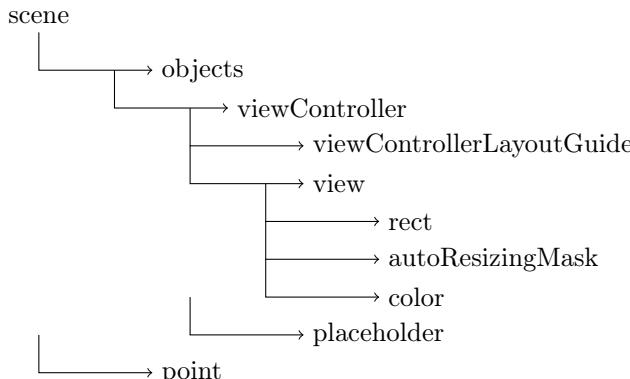
Reprezentarea sub formă de arbore a blocurilor XML este de forma:

```

1 <scene sceneID="YnV-Ty-V8r">
2   <objects>
3     <viewController id="HLP-Tz-EFz" ...>
4       <layoutGuides>
5         ...
6       </layoutGuides>
7       <view key="view" contentMode="scaleToFill" id="IAZ-eV-Kmg">
8         <rect key="frame" x="0.0" y="0.0" width="600"
9           " height="600"/>
10        <autoresizingMask key="autoresizingMask"
11          widthSizable="YES" heightSizable="YES"/>
12        <color key="backgroundColor" white="1" alpha
13          ="1" colorSpace="calibratedWhite"/>
14      </view>
15    </viewController>
16    <placeholder .../>
17  </objects>
18  <point key="canvasLocation" x="4762" y="97"/>
19 </scene>

```

Figure 2.2: Scene tag



Un atribut important de menționat ar fi **id**, atribut prezent în aproape toate tagurile. Atributul **id** specifică un număr de ordine unic pentru fiecare entitate din storyboard, astfel fiecare relație sau conexiune se referă la acest id.

În momentul în care se adaugă o nouă scenă, se creează un tag copil **scene** în cadrul tagului **scenes** în storyboard. Tag-ul **point** specifică offsetul scenei în cadrul canvas-ului de lucru.

Tag-ul **viewController** reprezintă View Controller -ul principal. Reprezentarea sa xml se poate vedea în figura 2.3:

```

1 <viewController id="1C9CF5D3-1A5A-41C4-BC52-F3977FC9F164"
  customClass="ViewController" sceneMemberID="viewController">

```

Figure 2.3: ViewController tag

Fiecare View Controller conține o scenă principală și este reprezentată de tag-ul **view**, așa cum se vede în figura 2.6

Proprietățile unei scene sunt reprezentate, aşa cum se vede în xml-ul de mai sus, prin tagurile **rect**, **color**, iar atributul **key** specifică caracteristica pe care tag-ul curent o definește.

În momentul în care adaugă view-uri copil la view-ul părinte, aşa cum se vede în figura alăturată:

Figure 2.4 ,

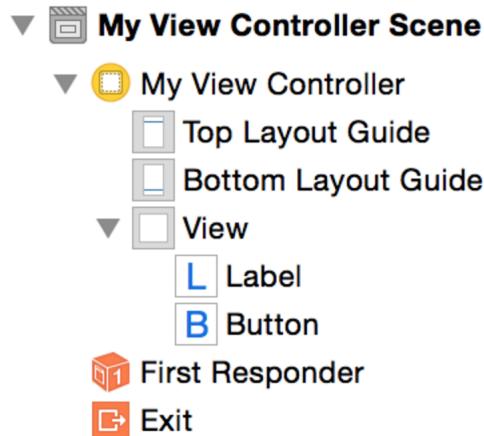


Figure 2.4: Views tag

reprezentarea xml asociată va fi (figura 2.6):

```

1 <view key="view" contentMode="scaleToFill" id="IAZ-eV-Kmg">
2 ...
3 <subviews>
4   <label opaque="NO" userInteractionEnabled="NO" contentMode="
5     left" text="Label" lineBreakMode="tailTruncation"
6     baselineAdjustment="alignBaselines" id="gWT-Ao-P2F">
7     <rect key="frame" x="119" y="0" width="87" height="98"/>
8     <color key="textColor" red="0.5" green="0.0" blue="0.31"
9       alpha="1" colorSpace="calibratedRGB"/>
10    <nil key="highlightedColor"/>
11  </label>
12  <button opaque="NO" contentMode="scaleToFill" fixedFrame="
13    YES" ... id="qJq-g4-c14">
14    <rect key="frame" x="75" y="86" width="46" height="24"/>
15    ...
16  </button>
17 </subviews>
18 </view>

```

Figure 2.5: View tag

Toate proprietățile setate pentru un **UIView**, oricare ar fi el - buton, text, imagine, etc. - se translatează automat în atribute sau taguri în fișierul xml.

În momentul în care un două view controlere se conectează cu un segue, un tag numit **connections** este creat și este inclus în interiorul tag-ului **viewController**.

Tag-ul **segues** reprezintă tranzițiile între view controlere sau butoane și alte view controlere.

```
1  <viewController id="qJq-g4-c14" customClass="MyViewController"  
2      sceneMemberID="viewController">  
3          ...  
4          <connections>  
5              <segue destination="gWt-Ao-P2F" kind="show"  
6                  identifier="NewSegue" id="IAZ-eV-Kmg"/>  
7          </connections>  
8      </viewController>
```

Figure 2.6: View tag

Tag-ul **segue** definește atributele **destination** și **kind**. Atributul **destination** specifică **id**-ul View Controllerului către care se duce tranzită, iar **kind** specifică tipul tranzitiei (show sau modal).

View Controllerele pot avea mai multe tranzitii și pot pointa către mai multe view controllere. Acest lucru este specificat prin multiple tag-uri **segue**.

2.1.1.1 Formatul xml

XML (Extensible Markup Language) este un limbaj care definește un set de reguli pentru codificarea documentelor într-un format atât human-readable, cât și machine-readable.

Scopul XML-ului este acela de furnizare de simplicitate, uzabilitate și generalitate în cadrul Internetului. XML-ul este un limbaj care este folosit în cadrul a numeroase aplicații. Aceasta a stat la baza protoocoalelor de comunicatie, cum ar fi XMPP. De asemenea, aplicațiile pentru Microsoft .NET Framework folosesc fișiere xml de configurare. În cadrul fișierului xd pe care îl vom studia în capitolul următor, metadata de configurare este stabilită tot prin limbajul xml.

Schema de translatare propusă în acest proiect a fost folosită și în cadrul acestor fișiere de configurare din xd.

2.2 Prezentare Adobe Experience Design

Adobe Experience Design este un nou utilitar pentru realizarea designului și prototipului de aplicații dedicate UX. Adobe XD este o soluție all-in-one care îi permite unui dezvoltator de iOS să creeze aplicații mobile și website-uri.

2.2.1 Structura unui fișier .xd

În urma realizării unei aplicații Adobe Experience Design, se creează un fișier cu extensia .xd, care va conține reprezentarea elementelor vizuale. Un fișier xd este realizat prin comprimarea mai multor fișiere de metadata cu extensia ".agc", prin realizarea operației de zip. Fișierele cu extensia .agc conțin informații legate de scenele din XD în format json.

Ierarhia de fișiere din cadrul unui design .xd (în urma aplicării comenzii unzip) este de forma:

Fișierul **metadata.xml** conține metadata referitoare la fișierul .xd - data de creare, de modificare, și id-urile canvas-ului.

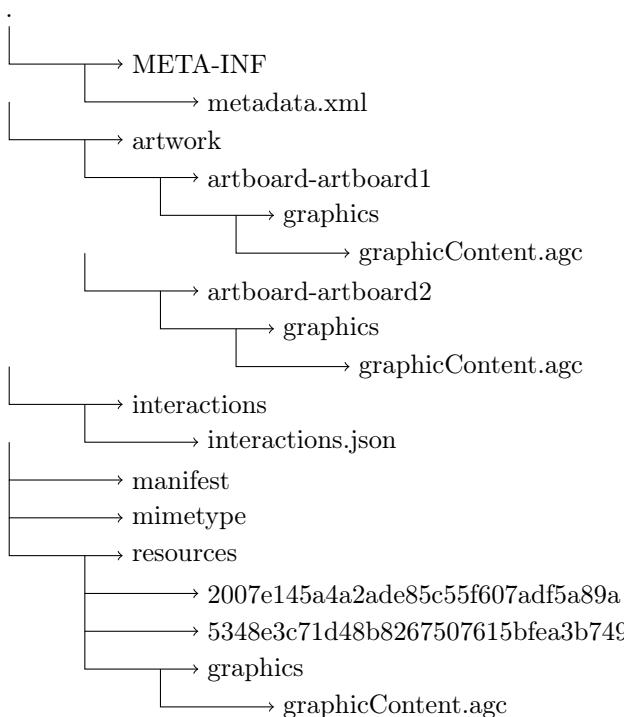


Figure 2.7: Ierarhie fișiere

Directorul **artwork** conține o serie de fișiere cu informații în format json despre elementele vizuale ale fiecărei scene¹ în parte.

Directorul **resources** conține o serie de imagini folosite în designul XD, salvate local sub numele id-ului corespunzător. Deasmenea, în cadrul acestui director se află informații legate de offset-urile și dimensiunile scenelor.

Directorul **interactions** conține informații legate de tranzițiile dintre scene.

Fișierul **manifest** conține metadata referitoare la ierarhia de componente grafice din cadrul fișierului XD.

Fișierul **mimetype** definește aplicația corespunzătoare XD-ului.

Un fișier de descriere a unei scene este reprezentat printr-o serie de proprietăți: **version** - specifică versiunea agc-ului, **resources**, respectiv **artboards** fac legătura cu restul fișierelor de descriere. Proprietatea **children** descrie lista de elemente vizuale ale scenei, precizând, astfel, pentru fiecare entitate, caracteristicile specifice, cadranul de încadrare și offsetul față de ierarhia de elemente față de care depinde.

2.2.1.1 Formatul json

JSON (numit și JavaScript Object Notation) este un format care folosește text human-readable pentru transmiterea obiectelor formate din perechi <cheie, valoare>. De aceea formatul JSON se poate serializa și deserializa ușor în structura de date dicționar.

Un dicționar declară programatic obiectul care realizează asociere între chei și valori. O pereche cheie-valoare din dicționar se numește un "entry". În interiorul dicționarului, cheile sunt

¹Se vor folosi termenii scene și artboard interschimbabil

stringuri unice, nenule. Tipul valorii nu are restricții, în afară de imposibilitatea declarării unei valori nule.

2.3 Translatarea și Scalarea dimensiunilor elementelor vizuale

În momentul în care se face translatarea dintre XD - XCode, este nevoie de aplicarea unor operații de scalare și translatare a dimensiunilor și a punctelor $\langle x, y \rangle$ ale tuturor elementelor vizuale. Aceste operații sunt dependente de dimensiunile "height" și "width" ale scenelor din XD, respectiv din XCode.

În XD, în mod implicit, scena default este cea de iPhone 6, cu proprietățile width=376, iar height=667, însă se pot alege diferite scene (iPhone 5/SE, iPad, Watch, Android Tablet, etc.) în funcție de tagetul dorit.

În XCode, în mod implicit, scena default este o scenă de format pătratic, în care width=height=600. Similar XD-ului, se pot crea instanțe de scene pentru un target specific (iPhone 5/SE, iPad, Watch, Android Tablet, etc.). În XCode, scena default are o caracteristică specială: se poate adapta oricărui tip de target, păstrând distanțele relative dintre obiecte. Această proprietate, însă, este dependentă de definirea unor constrângeri între elemente. În acest caz, în momentul în care un asset își schimbă poziția sau dimensiunea, atât el, cât și elementele vecine își vor modifica dimensiunile și pozițiile relativ la acesta. De exemplu, se poate centra o imagine orizontal într-o scenă din storyboard. Acest comportament este definit de reprezentarea xml din figura 2.8. Pe măsură ce user-ul rotește device-ul iOS, imaginea rămâne centrată orizontal, atât în formatul landscape, cât și în formatul portret.

XCode-ul definește o constrângere sub forma unei relații matematice. În figura 2.8, se indică faptul că imaginea curentă va avea următoarele constrângeri: lățimea sa va fi de 254, lungimea de 414, și se va centra orizontal în cadrul scenei.

```

1  <imageView ...width="414" height="254"/>
2      <constraints>
3          <constraint firstAttribute="width" constant="414" id
4              ="Bcj-uk-T1b"/>
5          <constraint firstAttribute="height" constant="254"
6              id="hiv-8m-nxo"/>
7      </constraints>
8  </imageView>
...
8  <constraint ... secondAttribute="centerX" id="3xj-be-ikQ"/>

```

Figure 2.8: Constraints

Așadar, pentru realizarea conversiei XD-XCode, respectiv XD-Xcode, avem două variante. În primul rând, ne putem folosi de Auto Layout din cadrul XCode și de scena default. Acest lucru ar implica, așadar, inserarea unor noi nivele computaționale pentru calcularea constrângерilor, dar ar oferi posibilitatea adaptabilității aplicației pentru mai multe target-uri. Cea de-a două variantă ar presupune construirea aplicației XCode specializate pe un target specific.

Soluția prezentă oferă suport pentru a două variantă, eliminând astfel construirea unor constrângeri între elemente. Cu toate acestea, soluția se poate extinde și în direcția suportului pentru Auto-Layout, prin extinderea schemei de translație - prin adăugarea unor funcții de prelucrare a constrângерilor.

Pentru implementarea celei de-a doua variante, este nevoie de obținerea gradului de scalare în momentul translatării, dar și a offsetului scenelor. Valoarea de scalare (atât pentru translatarea XD-Xcode, cât și pentru XCode-XD) se poate obține în urma formulelor:

$$xAxisScaleFactor = xAxisInitialArtboard/xAxisFinalArtboard$$

$$yAxisScaleFactor = yAxisInitialArtboard/yAxisFinalArtboard$$

cu precizarea că: $xAxisScaleFactor$, $yAxisScaleFactor$ reprezintă valorile de scalare pe axe X, respectiv Y, $xAxisInitialArtboard$ și $yAxisInitialArtboard$ reprezintă lungimea, respectiv lățimea scenei initiale (de la care se face translatarea), iar $xAxisFinalArtboard$, $yAxisFinalArtboard$ reprezintă lungimea, respectiv lățimea scenei finale (în care se face translatarea) Pentru obținerea lățimii și lungimii obiectelor vizuale, se aplică formulele:

$$width = initialWidth * xAxisScaleFactor$$

$$height = initialHeight * yAxisScaleFactor$$

cu precizarea că: $width$, $height$ reprezintă valorile finale ale lungimii, respectiv lățimii, $initialWidth$, $initialHeight$ reprezintă valorile initiale ale lungimii, respectiv lățimii elementelor vizuale.

în cazul obținerii valorilor x și y, trebuie să se ia cont de faptul că în XCode, offseturile elementelor vizuale sunt relative la elementele părinte. De exemplu, în cazul unui view cu punctul de offset $\langle x1, y2 \rangle$ și cu elementul vizual "Asset1" cu punctul de offset $\langle x2, y2 \rangle$, valoarea absolută a lui "Asset1" corespunde punctului $\langle x1 + x2, y1 + y2 \rangle$. În cazul soluției prezentate, view-urile vor ocupa întreaga suprafață a viewConrollerului, așadar punctele $\langle x1, y1 \rangle$ va fi echivalent cu punctul $\langle 0,0 \rangle$.

În cadrul reprezentării sub formă de agc, un element vizual este caracterizat de un offset cu valoare absolută pe suprafața de design. Pentru obținerea valorilor relative la scena de care aparțin, este nevoie de calcularea offseturilor pe baza grupurilor și a scenelor de care aparțin.

Așadar pentru calcularea valorilor x (pentru fiecare element vizual "X") în cazul translatăiei XD-XCode, formulele aferente sunt:

$$startXArtboard = startXArtboard - \sum xValue$$

$$translatedValue = (translatedValue - startXArtboard * xAxisScaleFactor);$$

cu precizarea că valoarea corespunzătoare lui y se calculează în mod similar, iar $startXArtboard$ reprezintă offsetul scenei din care face parte assetul "X", la care se scade valoarea tuturor valorilor "xValue" corespunzătoare grupurilor din care face parte assetul "X".

Calculul valorilor x, respectiv y în cazul translatăiei XCode-Xd se calculează astfel:

$$translatedValue = (translatedValue * xScaleFactor) + startXArtboard;$$

cu precizarea că valoarea corespunzătoare lui y se calculează în mod similar.

Valorile $startXArtboard$, $startYArtboard$, $xValue$, $yValue$ se calculează pe baza valorilor atributelor "tx", "ty" ale proprietății "transform" specifice scenelor, respectiv grupurilor.

2.3.0.1 Motivația prezentării celor două structuri

În acest capitol s-a prezentat structura și modul de reprezentare a celor două formate specifice XCode, respectiv XD. De asemenea, s-au precizat diferențele, dar și asociările care se pot construi între cele două formate. Aceste aspecte furnizează contextul în care soluția curentă rulează și vor ajuta la înțelegerea schemelor de translatare care urmează să fie explicate în capitolele următoare.

Chapter 3

Prezentarea schemelor de translatare

În acest capitol se va prezenta modul de translatare între cele două reprezentări XCode și XD sau, xml, respectiv json.

Formatul json este similar xml-ului prin faptul că amandouă reprezentările descriu structuri de date și obiecte serializabile. Multiple protocole bazate pe xml reprezintă aceleași structuri de date ca json-ul, în mod interschimbabil.

Găsirea unei scheme de translatare între cele două reprezentări se rezumă așadar la găsirea multiplelor mapări între tag-urile și atributele xml-ului, respectiv json-ului. De asemnea, anumite dependențe variază în funcție de aplicarea anumitor aplicații față de anumite atribute sau taguri.

Asocierile dintre tagurile xml și json se află într-o relație one-to-one, de aceea schema de translatare trebuie să fie bidirectională.

3.1 Translatarea din XCode în Adobe Experience Design

Importul elementelor vizuale din XCode în XD presupune parcurgerea fișierelor “storyboard” și inspectarea tagurilor cu atributele corespunzătoare. În urmă inspectării, trebuie să se obțină o reprezentare de tip json a fișierului de intrare. De aceea, schema de translatare pe care o propun este descrierea unui model de asociere între tagurile xml și elementele json, schemă reprezentată, la rândul său, de un fișier .json.

Această schemă va fi reprezentată într-un format json datorită caracteristicilor specifice acestui format: reprezentare human-readable, extensibilitate, serializabilitate, deserializabilitate rapidă în și dintr-un dicționar. Un astfel de dicționar asigură accesul la valori într-un timp constant.

Schema de translatare din xml în json a avut parte de două versiuni.

Prima versiune se bazează pe reprezentarea fiecărei proprietăți din json sub o formă ușor de extins și eficientă din punct de vedere computațional. În acest sens, m-am folosit de mapări <cheie, valoare> care să reprezinte proprietățile și valorile corespunzătoare.

Aceste mapări corespund unor serii de dicționare care reprezintă template-urile elementelor json (formatul în care vrem să translatăm). Așadar, schema propusă va fi construită dintr-un dicționar de template-uri specifice diferitelor proprietăți.

Fiecare cheie din acest dicționar principal reprezintă numele unei proprietăți din json, iar valoarea corespunzătoare - template-ul proprietății json.

Pe lângă acest dicționar - pe care o să îl numim **rootDict** - avem nevoie de o mapare între tagurile xml și proprietățile json care specifică același element vizual/ logic. Pentru aceasta, ne folosim de un dicționar de mapare suplimentar care va fi sub forma (<nume_tag_xml>, <nume_proprietate_json>). Pe acest dicționar îl vom numi în continuare **mapDict**.

Așadar, prin intermediul celor două dicționare rezolvăm problema mapării dintre cele două reprezentări. Apare, însă altă problemă: cum rezolvăm legăturile dintre proprietăți ?

Figure 3.1

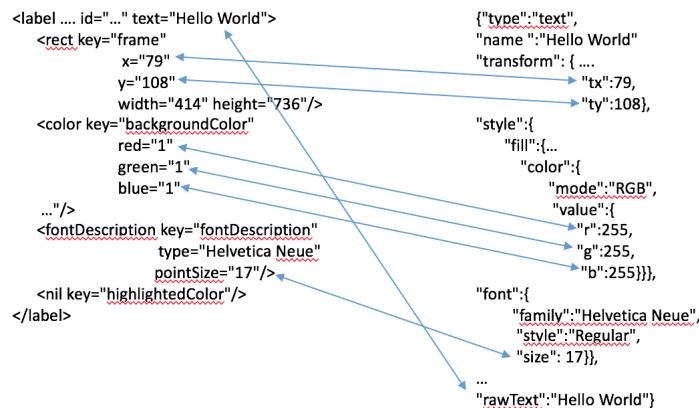


Figure 3.1: Switch tag

De exemplu, fie figura 3.1 care reprezintă formatul unui text în XCode, respectiv în XD și câteva asocieri dintre reprezentări. Proprietățile care variază - dimensiunile cadranului (**rect**), culorile (**color** cu atributele **fontDescription** și **background**), textul efectiv (atributul **text**) - vor trebui să fie înlocuite corespunzător în template-ul de json.

De aceea introducem niște notății care ne ajută în realizarea acestor asocieri.

Afirmăție 3.1. *Un string precedat de caracterul "\$" reprezintă o valoare care trebuie să fie înlocuită, conform comportamentului definit de acesta.*

Afirmăție 3.2. *Un string care are în compoziția sa caracterul ":" reprezintă nivelele de indirecțare într-o reprezentare json sau xml. Fiecare substring precedat de ":" reprezintă un nou nivel de indirecțare al unui format care poate fi transformat într-un dicționar - pe care îl vom numi **generalDict** .*

Afirmăție 3.3. *Fie string-ul "x1.x2.xn" care respectă afirmația 3.3. Împărțim stringul initial în mai multe substringuri delimitate de ":" și obținem un vector de substringuri de dimensiune n. Acest vector va avea următoarea proprietate: primele n-1 substringuri sunt chei în dicționarul **generalDict** prezentat în afirmația .*

Afirmăție 3.4. *Toate afirmațiile de mai sus sunt valabile și în cazul aplicării acestora simultan (pentru același string).*

Pentru a exemplifica afirmațiile de mai sus, ne vom folosi de figura 3.2 și de string-urile \$color.red, \$text.

Procedeul de aplicare al definițiilor este următorul:

- Pentru \$color.red (cheia "r"):

1. Stringul este precedat de caracterul "\$", așadar acesta va trebui să fie înlocuit, conform afirmației 3.1

```

1  {"textField" : {
2      "type" : "text",
3      "name" : "$text",
4      "transform" : {
5          "a" : "1",
6          ...
7          "tx" : "$rect.x",
8          "ty" : "$rect.y"
9      },
10     "style" : {
11         "fill" : {
12             "type" : "solid",
13             "color" : {
14                 "mode" : "RGB",
15                 "value" : {
16                     "r" : "$color.red",
17                     "g" : "$color.green",
18                     "b" : "$color.blue"
19                 ...
20             }
21         }
22     }
23 }

```

Figure 3.2: Schemă translatare xcode->xd

2. Evaluam valoarea stringului, conform afirmației 3.3
3. Împărțim stringul color.red într-un vector de substringuri delimitate de “.” => generalDict = [color, red] unde n = |generalDict| = 2
4. Vectorul specifică faptul că primele n-1 substringuri din generalDict vor fi chei, iar ultimul substring va reprezenta valoare (conform afirmației 3.3); Așadar, \$color.red va fi înlocuit cu valoarea atributului “red” al tagului “color”.

Afirmație 3.5. *O cale reprezintă nivelele de indirectare din cadrul unui dicționar, relative la o valoare. Reprezentarea unei căi este realizată prin concatenarea cheilor în drumul spre valoarea dorită, delimitate prin “.”.*

Afirmație 3.6. *Un dicționar “invers” se definește relativ la un alt dicționar. Fie dicționarul “invers” I, iar dicționarul principal P. Fie o valoare de forma “\$x1.x2.xn” într-un nivel oarecare np în cadrul P. Fie calea asociată acestei valori de forma “c1.c2.cn”. Atunci I va conține o valoare egală cu “c1.c2.cn”.*

Notă 3.1. *Construirea fișierului json final se realizează într-un dicționar pe care o îl numim resultDict.*

Am stabilit, așadar modul în care reprezentăm mapările, dar cum facem efectiv aceste asocieri? Prin folosirea unui dicționar “invers” care va fi stocat tot în **mapDict** așa cum se poate vedea în figura 3.3.

Dicționarul invers ne va ajuta să găsim, pentru un anumit tip de element xml, proprietățile care trebuie înlocuite și să facem această modificare.

Algoritmul pentru această metode este următorul:

1. Pentru a realiza o separare între cod și schemele de translatare, acestea vor fi stocate în fișiere separate, care la pornirea aplicației, vor fi încărcate în memorie.
2. Se parurge fișierul storyboard asociat aplicației XCode prin intermediul clasei NSXML-Parser. Pe măsură ce se parurge fișierul, obținem numele tagului curent și un dicționar cu atributele și valorile asociate.

```

1  {"imageView." : "style.fill.pattern.href",
2   "rect." : {
3     "x" : "textField.transform.tx",
4     "y" : "textField.transform.ty",
5     "width" : {
6       "rect" : "rect.shape.width",
7       "text" : "textField.shape.width",
8       "shape" : "imageView.style.fill.pattern.width"
9     },
10    "height" : {
11      "text" : "textField.shape.height",
12      "rect" : "rect.shape.height",
13      ...
14    }
15  }

```

Figure 3.3: Schemă translatare mapDict

3. Verificăm dacă există vreo numele tagului curent apără în dicționarul mapDict

- **Cazul Adevarat:** Valoarea mapării găsite reprezintă denumirea proprietății în XD. Fie această valoare X. Se găsește entry-ul din dicționarul rootDict care are cheia egală cu X. Valoarea entry-ului va reprezenta template-ul elementului vizual în reprezentarea XD și va fi adăugat la rootDict.
- **Cazul Fals:** Tagul curent reprezintă o caracteristică de la care trebuie să preluăm anumite informații. De aceea, căutăm în dicționarul invers numele tagului și obținem o valoare X. Valoare respectivă poate fi de două tipuri:
 - string : caz general, calea către obiectul care trebuie înlocuit este similară pentru orice tag. Se parcurge calea definită sub forma "c1.c2.cn" (conform dafirmării 3.5) din dicționarul rootDict; Se obține o valoare Y sub forma "\$x1.x2.xn"; În continuare se procedează similar algoritmului din 3.1.
 - dicționar :

Cea de-a doua metodă de translatare va explica ulterior, după translatarea din XD în XCode, din cauza unor noțiuni specificate în cadrul aceluia capitol.

În urma obținerii dicționarului json, există două posibilități. În primul rând, dicționarul obținut se poate copia în Clipboard (prin intermediul clasei NSPasteboard), urmând ca apoi, în urma unei acțiuni de Cmd + V în scena XD, să se copieze elementele vizuale în fișierul de design.

A doua modalitate este reprezentată de construirea ierarhiei de fișiere prin împărțirea fiecărei scene într-un fișier separat. De asemenea trebuie generate automat id-uri unice în cadrul aplicației pentru fiecare element, fie el logic, sau vizual. Fișierul manifest va conține ierarhia de fișiere, descrisă în json, extinsă prin caracteristici precum id unic, cale către fișiere din ierarhie și dimensiune.

3.1.1 Translatarea Switch-ului și a altor elemente grafice similare

Un switch permite utilizatorului pornirea sau oprirea rapidă a unor opțiuni mutual exclusive. Interface Builder are un obiect de tip Switch care poate fi ușor adăugat aplicației. XD-ul, însă, nu are un element de sine stătător care să reprezinte un astfel de element vizual. Există librării

care au construite anumite elemente vizuale (cum ar fi butonul, slider-ul, etc.) dar care descriu segmentele din care este alcătuit assetul.

De aceea, pentru translatarea acestor obiecte, se creează un fișier care conține template-ul obiectului respectiv din XD. Pentru modificarea caracteristicilor (dimensiuni cadran, culoare, etc.), se vor folosi schemele de translație, similar algoritmului prezentat anterior.

3.1.2 Translatarea Interacțiunilor XCode->XD

O interacțiune în XCode, denumită "segue", este o conexiune care reprezintă tranzitia dintre o scenă A și o altă scenă B. Relațiile și conexiunile sunt reprezentate vizual printr-o săgeată de la scena expeditor, la scena destinație. În mijlocul acestei săgeți se află o formă care specifică tipul de segue. Tipurile de segue se pot vedea în figura 3.4. Soluția propusă are suport doar pentru Show segue.

Figure 3.4

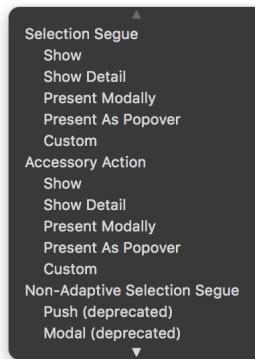


Figure 3.4: Switch tag

O interacțiune între un element vizual și o scenă se realizează prin intermediul id-urilor unice, specifice fiecărui asset. Acest lucru este valabil atât în XCode, cât și în XD. De aceea este nevoie o mapare între un id și elementul vizual pe care îl specifică.

Astfel, soluția propusă implementează un dicționar cu mapări `<id_unic, array>` (pe care îl vom numi `segueDict`), unde array reprezintă un vector de două obiecte. Primul obiect este reprezentat de numărul scenei din care face parte assetul, iar al doilea - numărul de ordine al elementului vizual din scenă.

De asemenea, trebuie reținut ultimul id întâlnit. Așa cum se vede în figura ??, interacțiunea depinde de id-ul destinație (tr7-0U-dbu) și de id-ul elementului vizual curent (Z5v-SE-Hvj). Acesată mapare se va adăuga unui dicționar (pe care îl vom numi `segueDict`) care va conține toate interacțiunile de pe canvas.

Pentru construirea interacțiunilor în XD, toate id-urile stocate în dicționarul `segueDict` trebuie să fie adăugate elementelor vizuale corespunzătoare, conform regulilor de mapare din dicționarul `segueDict`.

Mapările din `segueDict` vor fi copiate în cele din urmă în fișierul `interactions.json`.

3.2 Translatarea din Adobe Experience Design în XCode

Exportul elementelor vizuale din Adobe XD în Xcode presupune parcurgerea fisierelor “agc” și inspectarea elementelor acestuia. Particularizarea problemei presupune găsirea schemei de translatare de la un format de tip json la un format de tip xml. Așa cum s-a precizat în secțiunea anterioară, maparea XD-Xcode este o mapare complexă de tip one-to-one, care se bazează pe moștenire și aplicare de operații.

Schema propusă pentru export se poate extinde pentru translatarea dintre json și xml pentru orice fel de aplicație, prin respectarea câtorva reguli explicite mai jos.

Pentru implementarea acestei scheme se vor folosi două dicționare cu mapari ale celor două limbi. În primul rând, anumite elemente json sunt în relație de echivalentă cu anumite taguri xml.

Afirmăție 3.7. *Numim relație de echivalentă, relația în care cele două entități din json, respectiv din xml reprezintă același obiect vizual (sau logic).*

Notă 3.2. *Tipul oricărui element din XD este specificat prin proprietatea “type”, care există în cadrul oricărei reprezentări json al unui asset. De exemplu, pentru text, tipul se numește tot “text”;*

Notă 3.3. *Proprietatea “type” din cadrul unei reprezentări nu specifică de una singură apartenența la o clasă logică de asset; În această categorie intră imaginile; Pentru a specifica apartenența la un anumit tip de asset trebuie îndeplinite o serie de reguli tipice categoriei respective.*

De exemplu, obiectul vizual “label” corespunzător reprezentării .xml conține ca asociere în .json un obiect caracterizat de tipul “text”. Aplicarea afirmației 3.3 se poate observa în figura 3.5 care va fi stocată într-un prim dicționar **defDict**.

Se poate observa faptul că acest dicționar conține o serie de mapări de tipul **<Key, Value>**, unde “Key” reprezintă valoarea specifică proprietății “type”. “Value” reprezintă fie un obiect de tip string, fie un obiect de tip dicționar. În cazul în care “Value” este un string, vom ști faptul că tipul obiectului curent este stabilit prin valoarea lui “Value”. În caz contrar, “Value” va reprezenta un alt dicționar ale cărui chei sunt stringuri de forma 3.3. Aceste chei menționate vor reprezenta niveluri de indirectare în dicționarul xd. Toate aceste chei vor reprezenta, la nivel logic, branchuri if-else, care ne ajută la descoperirea categoriei din care face parte assetul curent. În momentul în care, prin parcurgerea nivelor de indirectare specificate în cheie, vom ajunge la o valoare diferită de null, algoritmul se va aplica pentru noul dicționar obținut. În cele din urmă vom obține categoria din care face parte assetul curent.

Notă 3.4. *Pentru o a numiță categorie de asseturi (de ex. pentru shape), pot exista mai multe modele de reprezentare, sau mai multe tipuri de asociere, În funcție de fișierul .xml.*

În cazul în care tipul obținut în urma aplicării algoritmului menționat mai devreme, se obține un string de forma afirmației 3.1, atunci acel tip corespunde notei 3.4.

De aceea, trebuie să păstrăm și această mapare în dicționarul din figura 3.5.

Cel de-al doilea dicționar va conține un template al reprezentării în care se va dori translatarea. Acest template este scris într-un format .json, datorită modului facil de reprezentare și extindere. De asemenea, un astfel de format poate fi deserializat într-un dicționar.

Organizarea acestei scheme de translatare .json este realizată după modelul din Figura 3.6. Tagurile din xml se pot împărtăși în trei categorii, în funcție de elementul reprezentat. Categoriile specificate sunt următoarele: content, subViews, subTags. Prin specificarea acestora, se dorește modularizarea implementării și separarea elementelor logice din cadrul proiectului. Fiecare tag, indiferent de categoria în care se află, se va conforma formatului prezentat în Figura 3.6.

O mapare corectă a unui tag trebuie să corespundă anumitor reguli de compunere și de afișare. De asemenea, construcția unui tag trebuie, în anumite cazuri, să respecte o anumită ordine. Acest lucru este necesar întrucât, în anumite cazuri, regulile de translatare între cele două

```

1  { ...
2  "text"      : "label",
3  "shape"      : {
4      "shape.type.rect" : {
5          "style.fill.type.pattern" : "imageView",
6          "style.fill.type.solid"   : "$frame"
7      },
8      "shape.type.path" : {
9          "style.fill.type.solid" : "$imageView"
10     },
11      "shape.type.line" : {
12          "style.fill.type.none" : "$lineEdit"
13      }
14  ...

```

Figure 3.5: XD2XCode Defs

reprezentări se bazează pe “moștenire”. Această relație garantează faptul că un tag care apare la un nivel inferior, va moșteni atributele părintelui, aka atributele copilului sunt “relative” la atributele părintelui. O bună exemplificare a acestei proprietăți este “moștenirea” cadrului în care se află un părinte. Tag-ul copil (tag care se află la un nivel inferior față de tag-ul de bază) va avea offsetul relativ la cadrul părintelui, iar dimensiunea în care “copilul” va putea fi așezat este dimensiunea spațiului “parinte”. Aceste specificații sunt reprezentate prin adăugarea elementului “order” cu lista de ordine corespunzătoare.

Dependențele și regulile de mapare vor fi specificate în cadrul elementului json “rules”. Acesta va conține un dicționar cu mapari <Key, Value>, unde Key reprezintă tagul/atributul care va fi modificat, iar Value va reprezenta la rândul său un dicționar de mapari <KeyDep, ValueDep>. “KeyDep” va reprezenta dependența tagului Key față de valorile “KeyDep”. Mapările explicate sunt specificate în cadrul ValueDep prin mapari de tipul <key_of_tag, value_from_KeyDep>. Parsarea și, în cele din urmă afișarea reprezentării în format “xml” va fi specificată prin intermediul tag-ului “toString”.

O proprietate din acest dicționar trebuie să conțină, la rândul său alte 4 proprietăți: **header**, **rules**, **toString** și, optional, **order**.

Header va conține toate atributele specifice tag-ului xml în care va fi realizată translatația.

Proprietatea **toString** va conține un vector de chei care va specifica ordinea de translatare și de afișare a atributelor unui tag xml.

Proprietatea **rules** specifică regulile care trebuie îndeplinite pentru o translatare corectă a elementului vizual curent. Această proprietate corespunde unui dicționar cu entry-uri de forma <regulă, modificador>. O regulă poate specifica două tipuri de stringuri:

- o caracteristică din cadrul headerului, specificată prin forma “\$header.<nume_atribut>”; acest lucru va sugera faptul că <nume_atribut> trebuie înlocuit conform valorii din modificador.
- un nume de copil tag; în acest caz obiectul **modificador** este un dicționar cu entr-uri de forma <dependentă, <dicționar de asociere>.

O altă proprietate, care este optională, este proprietatea “default” care se întâlnește la proprietățile care aparțin clasei de asset-uri “color”, “fontDescription”. În momentul în care, aceste proprietăți lipsesc din reprezentarea xml, în momentul în care se reiazează translatarea, aceste proprietăți vor primi valorile implicate - specificate prin dicționarul “default”.

```

1  {
2   "view" : {
3     "header": {
4       "key" : "view",
5       ...
6       "toString" : ["key", "contentMode", "id"] },
7       "toString" : ["header", "rules"],
8       "rules" : {
9         "$header.id" : "$rand",
10        ...
11        "subviews" : {
12          "$children.$sceneNo.artboard.children" : {}},
13        "color" : {
14          "$children.$sceneNo.style.fill.color.value" : {
15            "red" : "$r",
16            ...
17            "colorSpace" :"calibratedRGB" }},
18        "toString" : ["rect", "autoresizingMask", "subviews", "color"],
19        "order" : ["$header.id", "rect", "autoresizingMask", "color", "subviews"] }
20      .... }

```

Figure 3.6: XD2XCode Schema

În anumite cazuri, pentru translatarea dintr-un format în altul, este nevoie de defininirea unor comportamente specifice. De exemplu, în cazul în obținerei numărului de linii al unui element vizual de tip "text area", este nevoie de obținerea a două valori din formatul agc. Prelucrarea acestor valori se realizează prin intermediul unei funcții, care primește ca parametrii cele două valori. Precizarea acestor comportamente, la nivel de schemă de translatare se realizează prin string-uri de forma "\$<COMPORTAMENT> <param1> ... <paramN>".

3.2.1 Translatarea Path-urilor și a diverselor figură geometrice

Pe măsură ce se desenează în cadrul unei scene se creează o linie numită **path**, care este compusă din unul sau mai multe segmente drepte sau curbe care pot fi manipulat cu precizie ridicată.

Un path este reprezentat în XD sub forma unei proprietăți json sub forma (figura 3.7):

Informațiile oferite de această reprezentare se pot translata ușor într-un **svg**.

Scalable Vector Graphics sau SVG este un format de reprezentare al imaginilor vectoriale 2D bazat pe xml. Reprezentarea internă a unei imagini svg se poate vedea în figura 3.8.

Pentru translatarea din formatul json în formatul xml este nevoie de substituirea proprietăților **viewBox**, **fill**, **stroke**, **d** din **path** cu caracteristicile specificate în schema json (Figura 3.9)

ViewBox reprezintă cadrul în care path-ul este vizibil. Aceasta va fi construit prin găsirea maximului și al minimului pe axe X, respectiv y. Valorile de minim vor reprezenta primele valori din **viewBox** (offsetul de pornire), iar diferența între minim și maxim pe axe x și y vor reprezenta lățimea și lungimea cadranelui.

Valoarea **path** va conține reprezentarea segmentelor din care forma geometrică este constituită.

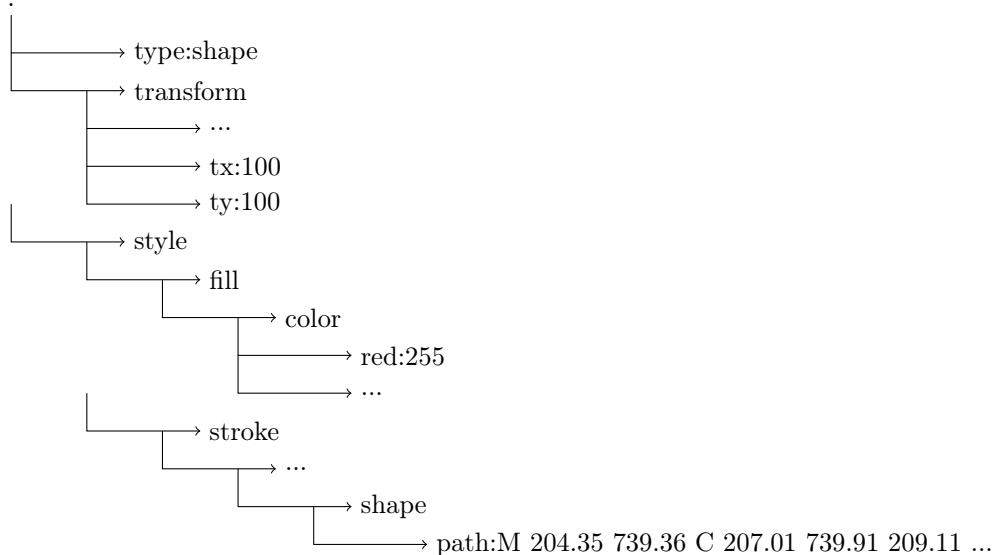


Figure 3.7: Atributul path

```

1 <svg xmlns="http://www.w3.org/2000/svg" viewBox="447.582 176.598
  87.418 92.402">
2 <defs>
3 <style>
4   .cls-1 {
5     fill: #fff;
6     stroke: #95989a;
7     stroke-width: 1px;
8   }
9 </style>
10 </defs>
11   <path id="rectangle-1" class="cls-1" d="M 204.35 ...
  translate(448 177)"/>
12 </svg>
  
```

Figure 3.8: Svg template

Translatarea unui path în format svg se realizează prin intermediul schemei XCode-Xd prezentată anterior. În figura 3.9 se specifică un comportament particular prin intermediul string-ului \$PATH. La nivel de implementare acest comportament particular este specificat printr-o funcție de prelucrare în funcție de argumentele specificate în schemă. Valorile obținute din procesarea reprezentării json din figură vor fi înlocuite într-un fișier svg template, conform proprietăților pe care acestea le definesc.

```

1 ...
2 "rules" : {
3     ...
4     "$header.image" : "$PATH $shape.path $name $style.fill.type
5         $style.fill.color.value $transform $style.stroke.color.value
6         $style.stroke.width",
7     "rect" : {
8         "$transform" : {
9             "key" : "frame",
10            "x" : "$pathx",
11            "y" : "$pathy",
12            "width" : "$path_width",
13            "height" : "$path_height"
14        }
15    ...

```

Figure 3.9: Path export

În urma translatării, se va obține un fișier svg cu elementul vizual regăsit în xd.

Din păcate, XCode-ul nu suportă formatul svg, de aceea pentru aducerea acestor asset-uri în aplicația XCode dezvoltată, mai este nevoie de un pas. Pasul constă în transformarea svg-ului într-un format png, suportat de XCode. Această transformare va fi realizată prin comanda convert din cadrul utilitarului ImageMagick.

Apelul comenții **convert** presupune crearea unui NSTask - un subprogram cu imaginea executabilă, în cazul de față, **convert** care va fi monitorizat de aplicația curentă.

Numele fișierului png rezultat în urma comenții ImageMagick se va compune din valoarea proprietății "name" și o valoare sha corespunzătoare obiectului path. S-a ales acest tip de denumire din următoarea cauză: "name" nu este o valoare unică - pot exista mai multe path-uri cu aceeași denumire. Întrucât fiecare de aceea, pentru diferențierea path-urilor se va adăuga un id unic. O primă variantă ar fi folosirea id-ului unic generat, însă în cazul în care s-ar realiza translatări multiple XCode-XD-XCode, s-ar genera o serie de imagini cu același conținut. De aceea, id-ul unic va fi reprezentat din valoarea SHA a obiectului "path".

Pentru a asigura conversia dimensiunilor și păstrarea proporțiilor din XD în XCode a imaginilor, aplicăm următoarea formulă:

- Se calculează valoarea pentru scalarea elementelor pe axe X, respectiv Y.
- Se aplică minimul dintre cele două valori obținute anterior.

Similar, se poate construi fișierul svg pentru linii prin adăugarea tag-ului <line ...> în cadrul svg-ului.

Pentru toate elementele convertite din svg în png, se transmit o serie de parametrii comenții **convert**, cum ar fi valoarea "none" pentru "background". De asemenea, culorile se vor specifica

și prin linia de comandă pentru a asigura conversia corectă a tuturor parametrilor path-ului inițial.

3.2.2 Translatarea textului

În XD, există două tipuri de text: text area (un cadran de o dimensiune variabilă în care se va pune text) și text simplu (cadranul e fix, relativ la textul pe care îl încadrează). Modalitatea de translatare a celor două tipuri de text este diferită din cauza modului de așezare și de reprezentare.

În cadrul elementelor de tip "text area", pentru calcularea cadranului este nevoie de offseturile "x", respectiv "y" și de dimensiunile "width" și "height" specificate direct în proprietățile agc-ului corespunzător. Însă, în cazul elementelor de tip "text" simplu, este nevoie de câteva procesări suplimentare. Un alt element specific textului în XD este modalitatea de reprezentare a numărului de caractere pe linie. Acest lucru se poate observa în figura: 3.10. Prin intermediul atributului "paragraphs" se obține numărul de linii, iar numărul maxim de caractere se obține prin atributul "lines" (prin formula MAX(lines.to - lines.from)). De asemenea, este nevoie de cunoașterea tipului și dimensiunii fontului pentru calcularea dreptunghiului în care se încadrează textul dat.

```

1 "paragraphs": [
2   {"lines": [
3     {"y": 0, "from": 0, "to": 6, "x": 0}
4   ]},
5   {"lines": [
6     {"y": 24, "from": 7, "to": 12, "x": 0} ] }
7 ]

```

Figure 3.10: Paragraphs

Elementele vizuale de tip "text", în forma de reprezentare agc, sunt caracterizate prin perechea $\langle x, y \rangle$ pentru a specifica offsetul în cadrul artboardului. Valoarea $\langle x \rangle$ specifică valoarea pe axa X a punctului stânga-jos a cadranului textului. Valoarea $\langle y \rangle$ însă este reprezentată prin valoarea pe axa Y a punctului stânga-jos a cadranului, la care se adaugă dimensiunea fontului. De aceea, pentru translatarea în reprezentarea xml echivalentă, se realizează calcularea punctului $\langle x, y \rangle$ conform specificațiilor de mai sus.

3.2.3 Translatarea grupurilor

Un grup în XD reprezintă o organizare logică a mai multor asset-uri într-o singură entitate. În urma acestei conversii, asseturile se vor comporta ca o singură entitate. Se pot aplica operații de translatare sau rotire asupra tuturor elementelor interne.

Reprezentarea sub formă unui json al unui grup este specificată în figura 3.11.

Proprietatea "transform" specifică modul de translatare sau rotire a groupului. Toate elementele care aparțin grupului (dicționarele care aparțin listei "children") vor fi modificate conform valorilor lui transform.

În Xcode, noțiunea de group este inexistentă, dar se poate face o asociere grup (în XD) - view (în Xcode). Pentru construirea dimensiunilor view-ului, se poate realiza o parcurgere inversă de la elementele constitutive grupului, la dimensiunea grupului. Ideea acestui algoritm s-ar baza pe parcurgerea colecției de elemente din grup și calcularea offsetului minim și maxim pentru

```

1  {
2      "type": "group",
3      "transform": {
4          "a": 1,
5          ...
6          "tx": 1647,
7          "ty": 40},
8      "group": {
9          "children": [
10             [...]]}

```

Figure 3.11: Group

ficare element. Cadranul grupului (view-ului din Xcode) va fi încadrat între offsetul minim al axelor x și y și offsetul maxim al acestora.

În XD, organizarea elementelor vizuale sub forma grupurilor duce la obținerea mai multor nivele de imbricare. De aceea, în urma translatării, în XCode se vor obține o serie de view-uri imbricate. În general, translatarea în mai multe view-uri imbricate nu corespunde, la nivel logic, cu organizarea la nivel de grup în XD. De asemenea, construirea implementării aplicației în Xcode, pe baza mai multor view-uri imbricate din Interface Builder, este destul de anevoiasă, și, posibil, nu corespunde cu cerințele aplicației. De aceea, am renunțat la asocierea grup-view. În acest sens, în momentul în care se întâlnește în reprezentarea agc un element de tip "group", se rețin valorile de offset, "tx" și "ty", continuându-se translatarea cu elementele care aparțin grupului. Cele două valori "tx" și "ty" ne ajută pentru obținerea valorilor absolute ale asseturilor corespunzătoare unui grup. De aceea, aceste valori se adaugă offsetului inițial al artboardului, așa cum a fost explicitat în secțiunea de scalare și translatare.

3.2.4 Translatarea imaginilor

Procesarea imaginilor se realizează similar procesării altor elemente vizuale, cu singura diferență că acestea trebuie importate fizic în aplicația XCode.

De aceea, este nevoie de obținerea căii absolute ale imaginilor care trebuie preluate. Obținerea căii absolute se realizează prin 3 metode:

- obținerea valorii pentru proprietatea "href", duce la obțierea căii către imaginea dorită
- Un fișier xd, după decompresie, așa cum s-a prezentat în figura 2.7 conține un director resources care, pe lângă fișierul de descriere al scenelor, conține și imaginile din XD. Numele imaginilor sunt reprezentate printr-un id unic, care se regăsește și în fișierul de descriere al XD-ului.

În urma obținerii căii absolute ale imaginilor, se face o copie locală a acestora într-un director "Resources" din cadrul proiectului XCode. Aceste fișiere vor trebui să fie adăugate manual (Drag Drop) în directorul "Supported Files" al aplicației XCode.

În cazul în care se realizează o serie de operații de translatare între XCode și XD (import, export, sincronizare), nu este nevoie de o copiere la fiecare astfel de operație, întrucât acest lucru ar presupune suprascrierea unor fișiere cu aceleasi informații. Acest lucru ar duce la un overhead destul de semnificativ, de aceea, înainte de realizarea copierii, se verifică existența fișierului în directorul "Resources".

3.2.5 Translatarea interacțiunilor

O interacțiune în XD reprezintă o legătură între un element vizual dintr-o scenă și o altă scenă. Prin interacțiuni, aplicația este animată și se pot realiza tranzitii între scene, în funcție de comportamentul descris.

Pentru a realiza translatarea interacțiunilor, se citește fișierul interaction.json, obținându-se astfel un dicționar cu mapări între id-uri - pe care o să în numim **segueDict**.

Următoarea etapă constă în parcurgerea dicționarului cu reprezentarea tutoror scenelor, și obținerea mapărilor între id și elementul vizual curent (mapări stocate într-un dicționar **mapDict**). În momentul în care se obține un id care există sub formă de cheie în dicționarul **segueDict**, se adaugă la **resultDict** - template-ul pentru obiectul de tip interacțiune.

Deoarece id-urile destinație pot reprezenta scene care încă nu au fost procesate, dicționarul final va trebui să fie parcurs încă o dată pentru înlocuirea id-urilor conform informațiilor descrise în **mapDict**.

Pentru realizarea corectă a translatărilor din XD în XCode, trebuie luată în considerare diferența majoră între tipurile de obiecte care suportă tranzitii. În Xcode, obiectele care suportă tranzitii sunt: **button** sau **view**, în timp ce în XD, nu există restricții de tip. De aceea, în momentul în care se realizează translatarea XCode - XD, inspectăm dicționarul **segueDict**. În momentul în care găsim un obiect cu id-ul prezent în dicționarul **segueDict**, obiectul respectiv va fi translatat într-un obiect xml de tip **button**.

3.3 Sincronizarea scenelor între cele două reprezentări

Sincronizarea se referă la modul în care elementele vizuale vor fi updateate în momentul în care se realizează o modificare în proiectul XD, sau în proiectul XCode. Voi incepe cu prezentarea a sincronizării de la XD și Xcode, sincronizarea inversă urmând să fie implementată ulterior.

3.3.1 Sincronizarea Adobe Experience Design - XCode

O variantă brută de implementare a sincronizării se poate constitui din două etape: monitorizarea fișierului de design și generarea tuturor scenelor, indiferent de elementele modificate. Generarea scenelor s-ar baza pe modulul de translatare XD-Xcode, însă această algoritm ar presupune un overhead destul de mare. În cazul în care fișierul de design se va modifica doar pentru câteva scene, regenerarea tuturor artboard-urilor ar fi costisitoare. De aceea, trebuie să se țină cont de obiectele modificate. Scenele care nu au fost modificate vor avea aceeași reprezentare în fișierul storyboard.

În continuare voi prezenta algoritmul de sincronizare care se bazează pe detecția modificărilor.

Implementarea sincronizării se bazează pe monitorizarea fișierului de design prin intermediul tool-ului Grand Central Dispatch (GCD) dispatch queues. **Dispatch queues** permit executarea unor blocuri de cod sincron, sau asincron, în funcție de modul de implementare. Acestea se pot folosi pentru rularea unor task-uri în thread separat față de aplicația principală.

În cazul de față, **task-ul** implementat se referă la monitorizarea unui fișier de design specificat prin calea sa absolută. În momentul în care un fișier XD este modificat, aplicația va fi notificată și va începe căutarea scenelor modificate. Această căutare se bazează pe o pre-salvare atât a offseturilor scenelor din Xcode cât și a sha-urilor corespunzătoare reprezentărilor agc ale acestora.

Offsetul este necesar deoarece un fișier Xcode conține toate scenele din storyboardul reprezentat. Scenele nemodificate sunt reprezentate prin scenele care au sha-ul în dicționarul de sha-uri presalvate. Maparea acestui sha este realizat împreună cu numărul de ordine al scenei corespunzător. În cazul în care o scenă s-a detectat că fiind modificată, se re-generează codul corespunzător (așa cum s-a prezentat în secțiunile anterioare). La fiecare nouă updatare, dicționarele cu offseturi și cu sha-uri trebuie refăcute.

Algoritmul de sincronizare rulează implicit în momentul în care se realizează translatarea din XCode în Adobe Experience Design. Pașii acestui algoritm sunt:

- Se creează translatarea XCode-XD prin conversia fișierului xml în format json. De asemenea, se construiește un vector cu $n+1$ offseturi, în care primele n valori reprezintă offseturile tag-urilor "scene" din cadrul fișierului .storyboard, iar ultimul offset, $n+1$, va reprezenta offsetul ultimul tag "/scene". Ultimul offset este necesar pentru obținerea dimensiunii ultime scene. Pentru înlocuirea scenelor pe baza modificărilor întâlnite, este necesară obținerea offsetului de început, dar și a dimensiunii în cadrul fișierului storyboard obținut prin formula:

$$\text{offsetList}[sceneNo + 1] - \text{offsetList}[sceneNo]$$

unde offsetList reprezintă lista de offseturi, iar $sceneNo$ numărul scenei curente. Se salvează sha-ul fiecărei scene translataate într-un dicționar "shaDict"

- Începe monitorizarea fișierului de design .xd. În momentul în care se realizează salvarea fișierului .xd, se realizează decompresia acestuia și se parcurge lista de fișiere din cadrul directorului "artworks". Se obțin valoarile sha ale fiecărui fișier destinat reprezentării scenelor.
- Se verifică apartenența fiecărui sha în lista de sha-uri presalvate (shaDict). Apar două situații:
 - Sha-ul se află în shaDict. În acest caz, scena curentă nu a fost modificată și reprezentarea în xml va fi aceeași.
 - Sha-ul nu se află în shaDict. În acest caz, scena curentă a fost modificată și se va realiza translatarea XD-XCode pentru aceasta.
- Se updatează fișierul storyboard, lista de offseturi și dicționarul de sha-uri pe baza scenelor modificate
- Algoritmul se reia pentru fiecare salvare a fișierului de design .xd.

Chapter 4

Implementare

În acest capitol voi oferi detalii despre arhitectură. De asemnea, voi prezenta o statistică a timpilor de rulare și modele de îmbunătățire ale aplicației.

4.1 Arhitectură

Managementul elementelor vizuale prezentat în capitolele precedente este implementat integral în Objective C.

Interacțiunea cu fișierele de design XD s-au realizat în două moduri: prin intermediul Clipboardului și prin crearea ierarhiei de fișiere xd.

În primă etapă, preluarea elementelor grafice din Adobe XD s-a realizat prin intermediul Clipboardului. Astfel, interacțiunea cu artboardurile din XD s-a realizat prin mecanismele de copy și paste ale scenelor. După translatarea din format XCode în format XD, rezultă un singur fișier .agc care va conține toate informațiile necesare creării fișierului de design. Conținutul acestui fișier va fi pus în Clipboard cu tipul **com.adobe.sparkler.design**, care va fi apoi copiat în fișierul xd.

Flow-ul importului prin intermediul Clipboardului se regăsește în figura 4.1, iar al exportului, prin intermediul Clipboardului, se regăsește în figura 4.2.

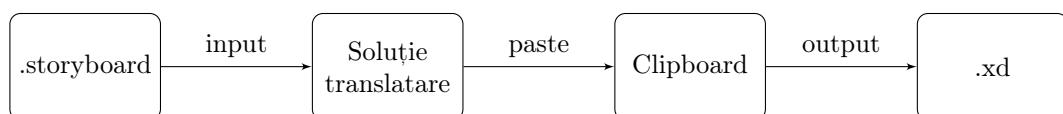


Figure 4.1: Clipboard Import

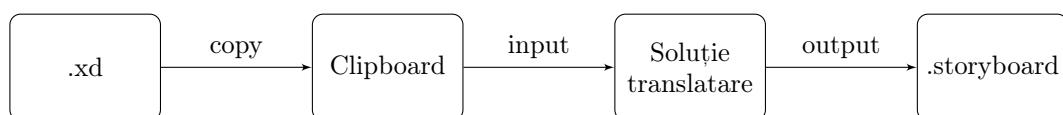


Figure 4.2: Clipboard Export

În a doua etapă - etapa finală - s-a renunțat la folosirea Clipboardului, în favoarea creării fișierului xd. Acest lucru s-a realizat din două motive: În primul rând, folosirea Clipboardului implică un flow mai greoi al aplicației și o interacțiune suplimentară a utilizatorului (overhead

adus de copy și paste). Cel de-al doilea motiv implică imposibilitatea copierii tranzităilor prin Clipboard.

Așadar, pentru preluarea informațiilor din cadrul unui fișier xd, se va realiza decompresia acestuia prin apelarea comenzi **zip**. În urma acestui apel, se vor obține ierarhia de fișiere așa cum a fost descrisă în capitolele 1, respectiv 2.

În urma translatării din XCode în XD, se va construi ierarhia de fișiere care va fi apoi comprimată în format .xd. Această comprimare se va realiza prin intermediul comenzi **zip -r -no-dir-entries <design xd> <ierarhie fișiere>**.

Arhitectura acestui model de lucru se regăsește în figurile 4.4, 4.3.

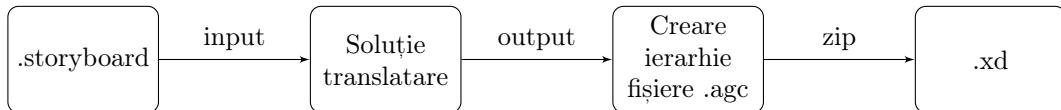


Figure 4.3: Simple Import

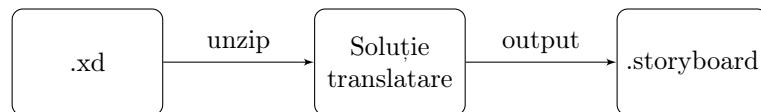


Figure 4.4: Simple Export

Soluția prezentată (sau BlackBox-ul prezentat în figura 4.6) se împarte într-o serie de module care implementează diferite concepe și funcționalități:

- Xml2Dict: realizează translatarea din formatul xml corespunzător fișierului .storyboard într-un format json. Subtagurile corespunzătoare tagurilor scene și subviews vor fi transformate într-un vector de scene, respectiv de elemente vizuale.
- Dict2Agc: se folosește de dicționarul json obținut de Xml2Dict și se folosește de o schemă de translatare XD - XCode pentru obținerea ierarhiei de fișiere xd;
- XCode2XD: implementează funcționalitatea de translatare din reprezentarea XCode în reprezentarea XD; la baza acestui modul se află schema de translatare xml->json;
- XD2XCode: implementează funcționalitatea de translatare din reprezentarea XD în reprezentarea XCode; la baza acestui modul se află schema de translatare json->xml;
- XDCreator: Creează ierarhia de fișiere .agc pentru construirea unui fișier de design xd
- Sync: implementează funcționalitatea de sincronizare între XD și XCode; monitorizează un fișier .xd, astfel încât orice modificare realizată în momentul salvării acestuia, să se revadă și în proiectul XCode asociat
- Helper: modul care implementează o serie de funcționalități generale necesare tuturor celorlalte module (ex. lucrul cu fișiere, crearea task-urilor pentru efectuarea de operații zip sau convert)

Acste module și interacțiunea dintre ele este reprezentată în figura 4.5.

În arhitectura prezentată, modulul XCode2XD poate fi interschimbat cu modulele Xml2Dict și Dict2Agc, întrucât aceasta implementează aceeași funcționalitate, dar prin scheme de translatare diferite.

O tehnologie folosită pentru testarea și implementarea cu ajutorul Clipboardului este Clipboard Viewer. Această aplicație arată conținutul oricărui clipboard, în funcție de tipul dorit. În

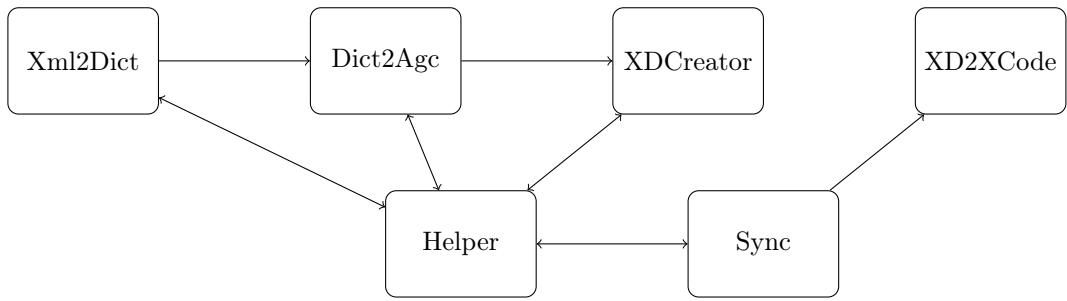


Figure 4.5: Arhitectură

cazul de față, pentru obținerea clipboardului specific aplicației Adobe XD, tipul clipboardului este reprezentat de "com.adobe.sparkler.design".

4.1.1 Demo

Aplicația este funcțională și implementează sincronizarea și translatarea directă și inversă a elementelor vizuale dintre Adobe XD și XCode. Realizarea proiectului a fost testată pe un sistem MacOSX, aplicația fiind scrisă în ObjC cu ajutorul IDE-ului XCode. Pentru rularea aplicației este necesar proiectul curent și, desigur, aplicația Adobe XD. Intern, proiectul se folosește de utilitarul ImageMagick pentru conversia între un format svg și un format png.

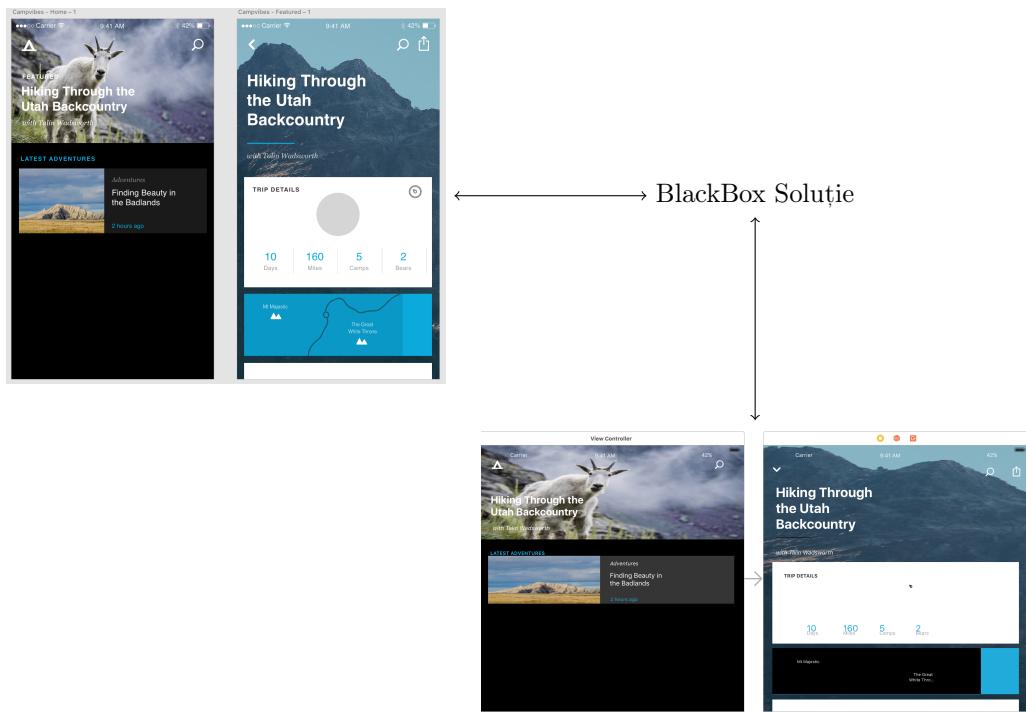


Figure 4.6: Flow-ul aplicației

4.2 Performanță

Realizarea translației este realizată în formatul .json din considerente de performanță. Un format json poate fi deserializat cu ușurință într-un dicționar imbricat, rezultând astfel o căutare rapidă a mapării dorite. Un astfel de dicționar asigură o complexitate de $O(nn)$, unde nn reprezintă numărul de niveluri de imbricare al dicționarului. În această aplicație, numărul de niveluri nu depășește 5.

În continuare voi prezenta graficele care specifică dependența între asset-uri și timpul de procesare al acestora.

Assets processed se referă la numărul de proprietăți (scene, elemente vizuale, dar și proprietăți -)

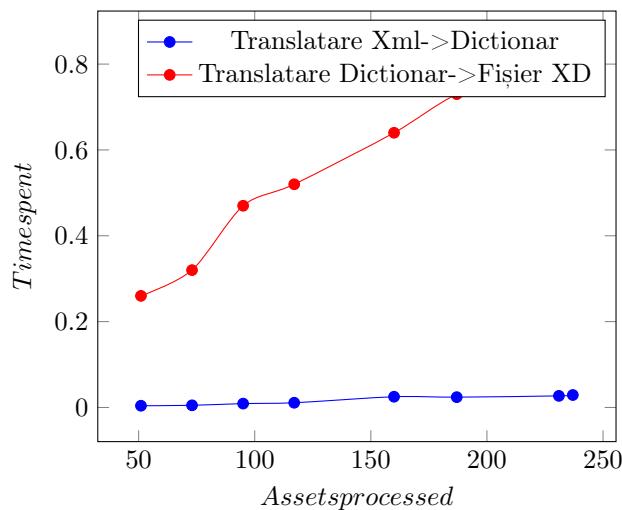


Figure 4.7: Grafic XCode->XD (Schema Translatare 2)

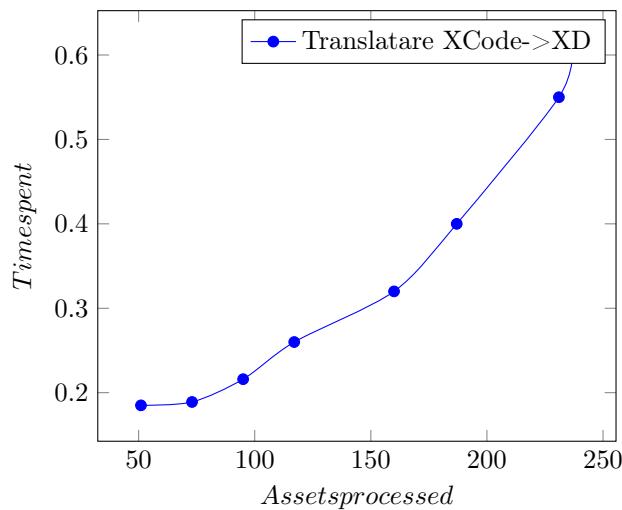


Figure 4.8: Grafic XD->XCode (Schema Translatare 2)

Se poate observa faptul că cele două scheme de translație XD->XCode, respectiv XCode->XD au tempi similari de rulare. Acest lucru era de așteptat, având în vedere că implementările se bazează pe același model de schema de translație. Un overhead se poate observa în cazul

translatării din XCode în XD, din cauza etapei intermediare de transformare a xml-ului într-un format json. De asemenea, în cazul translatării XCode->XD, apare o etapă computațională în plus față de translatarea XD->XCode. Această etapă constă în găsirea căii absolute a imaginilor care se regăsesc în aplicația curentă. O primă variantă a fost căutarea recursivă în directorul proiectului XCode, însă această abordare are două dezavantaje. În primul rând, există posibilitatea ca imaginile căutate să nu fie în directorul proiectului curent, iar în al doilea rând, aşa cum se vede în figura 4.9 (coloana trei), timpii de căutare să fie mult mai mari decât timpul propriu-zis de translatare. De aceea, se vor folosi informațiile din fișierul "project.pbxproj". Acest fișier se poate regăsi în orice proiect de tipul XCode și va conține meta-date legate de aplicația dezvoltată. Printre aceste metadate se află și informații cu privire la locația imaginilor folosite. Timpii de rulare pentru această metodă se regăsesc în tabelul 4.9 (coloana patru). Toți timpii de rulare din tabelul 4.9 sunt relativ la numărul de scene, respectiv numărul de proprietăți (coloanele cinci și sase). În prima coloană se specifică timpul de translatare dintr-un fișier xml (în cazul de față - storyboard-ul) într-un fișier json asociat.

Nr.	Xml->Json	Json->Xd(V1)	Json->Xd(V2)	Nr Scene	Nr Proprietăți
1	0.004	3.99	0.26	1	50
2	0.005	4.96	0.32	2	73
3	0.009	5.99	0.47	3	95
4	0.011	7.05	0.52	4	117
5	0.025	9.00	0.64	5	160
6	0.026	10.60	0.73	5	187
7	0.027	13.20	0.83	7	231

Figure 4.9: Tabel Import

În figura ?? se pot observa diferențele între timpii de rulare a celor două metode de sincronizare. În graficul de culoare roșie se reprezintă sincronizarea XD-XCode prin generarea tuturor scenelor, independent de numărul de scene modificate. Graficul de culoare albăstră specifică sincronizarea XD-Xcode, dependentă de scenele modificate. Se poate observa o diferență semnificativă a timpilor de rulare în cazul în care un număr mic de scene vor fi modificate. Axa X este reprezentată de numărul de scene modificate, iar axa Y este reprezentată de timpul de rulare corespunzător metodei de sincronizare ales. Cele două grafice au fost generate pentru un număr de scene = 6.

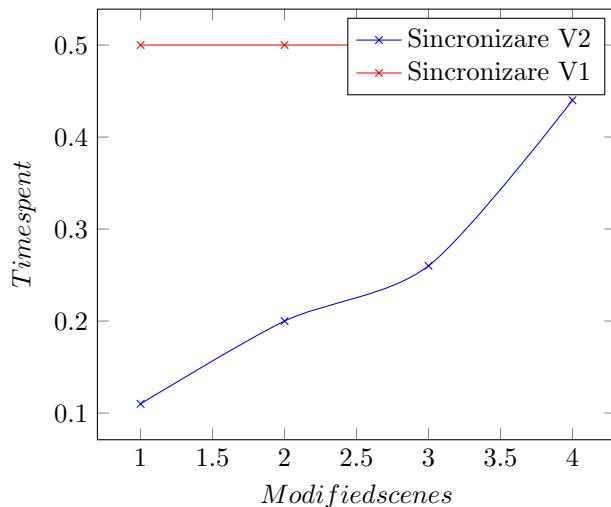


Figure 4.10: Grafic Sincronizare

4.2.1 Comparație Scheme de translatare

După cum se observă în descrierea procedeelor de translatare XCode - XD și XD - XCode, cele două scheme au metode de implementare destul de diferite.

Prima schemă de translatare XCode - XD se bazează pe parcurgerea fișierului de input și pe realizarea translației pe măsură ce se procesează fișierul. Pentru fiecare tag întâlnit se oferă un template care va fi procesat la rândul său pe baza regulilor definite.

Există însă o serie de dezavantaje. În primul rând, fiecare template al unui tag va fi parcurs de două ori (prima oară pentru copiere, iar a doua oară pentru înlocuirea anumitor proprietăți, conform schemei). Schema de translație nu poate însă afla categoria din care face parte un element vizual, analizând numai tagul curent. De aceea, pe baza tagului părinte curent se va genera un template, apoi pe măsură ce se observă alte elemente care indică faptul că tipul assetului curent este altul, se modifică și template-ul. Un exemplu al acestui caz este regăsit în cazul butoanelor. Un buton este reprezentat în următoarea formă:

```

1 <button opaque="NO" ... id="qj4-Ou-Uog">
2   <rect key="frame" x="68" y="189" width="212" height="120"/>
3   <state key="normal" title="Button" image="art2.png"/>
4 </button>

```

Figure 4.11: Tipuri de butoane specificate prin tag-ul state

Tag-ul copil **state** specifică faptul că butonul este de tip **text**, fie de tip **imagine**, în funcție de atributele prezente - **title**, respectiv **image**.

În momentul în care se ajunge la tag-ul **button**, se va construi un template json care specifică tipul - în mod default un buton de tip text. Dacă, însă, în momentul în care se ajunge la tag-ul copil **state**, iar atributul său corespunzător este **image**, tipul template-ului trebuie înlocuit.

Acest neajuns este eliminat prin schema de translație XD - XCode. Această schemă asigură obținerea tipului corect de la începutul procesării, datorită modului de procesare a tuturor caracteristicilor specifice unui element vizual. Așadar în cazul de față, această schemă de translatare ar asigura găsirea corectă a tipului assetului prin verificarea tag-ului parinte și a atributelor specifice tag-ului **state**. Assadar, pentru a afla tipul corect al butonului - de tip text sau imagine, se verifică prezența atributelor "title" sau "image". În cazul în care apar ambele atribută, în reprezentarea acă vor fi construite două elemente vizuale - unul de tip text, și altul de tip imagine.

În principiu, neajunsul specificat în prima schemă este datorat tipului de fișier de intrare xml, care permite procesarea unui singur tag la un moment dat. Complexitatea de procesare este de asemenea mai ridicată față de cea de-a doua schemă. Schema de translatare folosită pentru conversia de la XD la XCode se poate folosi și în cazul translatării XCode - XD, prin conversia mai întâi a fișierului de input din xml în json.

O altă diferență majoră dintre cele două scheme de translatare ar fi ușurința de extindere, dar și de înțelegere. Schema de translatare XD - XCode asigură extensibilitate, prin construirea template-ului, mapărilor și specificarea valorilor default într-o reprezentare ușoară și intuitivă. Specificarea unui comportament particularizat se poate construi prin adăugarea acestuia în cadrul schemii de translație și adăugarea unei funcții de implementare.

Chapter 5

Concluzii

În cadrul acestui proiect am prezentat o schemă de translatare între două sisteme de design, respectiv de implementare a aplicațiilor mobile. Aceste scheme prezentate se pot extinde pentru translatarea diferitelor formate (de ex. xml în json, json în xml). De asemenea, s-a prezentat o soluție de sincronizare a elementelor grafice între XD și XCode, la nivel de scenă. Sincronizarea inversă se poate realiza în același mod, prin implementarea algoritmului propus.

Această schemă de translatare se poate folosi în primul rând pentru managementul elementelor vizuale, aşa cum este prezentă soluția de față, dar se poate extinde pentru o serie de aplicații, datorită schemei generale de translatare.

5.1 Îmbunătățiri ulterioare

Această aplicație se poate extinde ușor prin transformarea schemelor de translatare. Câteva îmbunătățiri care se pot aduce acestei aplicații sunt:

1. Suport pentru multiple tipuri de controlere (Table View Controller, Collection View Controller, Navigation Controller, Tab Bar Controller, Page View Controller, GLKit View Controller sau Custom View Controller); Acest lucru se poate realiza prin extinderea schemei de translatare - definirea noilor mapări și dependențe;
2. Adăugare constrângeri pentru folosirea Auto Layout din cadrul Interface Builder
3. Suport pentru multiple tipuri de interacții (Show Detail etc.)
4. Translatarea multiplelor path-uri/figuri geometrice care aparțin aceluiași grup într-o singură imagine.
5. Reprezentarea vizuală a mapărilor - pentru extinderea facilă a aplicației
6. Permiterea sincronizării pentru multiple fișiere; Acest lucru se va poate realiza prin modificarea ierarhiei de fișiere create în interiorul aplicației XCode;
7. Translatarea elementelor view din XCode în elemente de tip grup în XD
8. Sincronizarea din XCode în XD, similar sincronizării inverse; Suport pentru apariția conflictelor (versionare)

Bibliography

- [1] XML Media Types, RFC 7303. Internet Engineering Task Force. July 2014.
- [2] Xcode on the Mac App Store. Apple Inc. Retrieved November 10, 2014
- [3] XML 1.0 Specification. World Wide Web Consortium. Retrieved 2010-08-22.
- [4] Media Type Registration for image/svg+xml. W3C. Retrieved 5 February 2014
- [5] Scalable Vector Graphics (SVG) 1.1 (Second Edition). W3C.
- [6] ECMA-262 (ISO/IEC 16262), ECMAScript® Language Specification, 3rd edition (December 1999)