Universitatea POLITEHNICA din București

Facultatea de Automatică și Calculatoare,
Departamentul de Calculatoare

LUCRARE DE DIPLOMĂ

Managementul elementelor vizuale pentru aplicații mobile

Conducător Științific:
Prof.dr.ing. Nicolae Țăpuș

Autor:
Claudia Rafaela Rogoz

București, 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 doreste realizarea unei scheme de management atât pentru aplicații mobile, cât și pentru site-uri web. Astfel, un developer de iOS va putea crea, atualiza și modifica entitățile grafice - imagini, text, scene, tranziții între ecrane - ale aplicațtiei iOS dezvoltate.

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

Contents

List of Figures

List of Tables

Chapter 1

Introducere

Proiectul de față este un plugin care vine în ajutorul developerilor și designerilor de aplicații user experience.

În capitolul acesta, se va realiza o scurtă descriere a proiectului, motivația și obiectivele aplicatiei, 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 printr-o translatare între diferite reprezentări.

O primă reprezentare este una logică, definită prin Xcode Interface Builder. Cea de-a doua reprezentare este definită prin produsul software Adobe Eperience 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

Finalitatea acestei aplicații este crearea automată a unei "punți" între două tehnologii dedicate designerilor de UX.

Așadar, principalul scop al acestui proiect este construirea unei scheme de translatare între cele doua reprezentări descrise anterior pentru a veni în ajutorul developerilor de iOS. Prin intermediul soluției descrise, crearea, sincronizarea, actualizarea, dar și colaborarea între membrii unei echipe de dezvolatare 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 translatare între cele două tehnologii pentru a fi folosit și în alte domenii de aplicații (baze de date).

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 realizare de prototipuri (Adobe XD). Totodată, prin intermediul acestui proiect se dorește construirea automată a aplicației XCode,

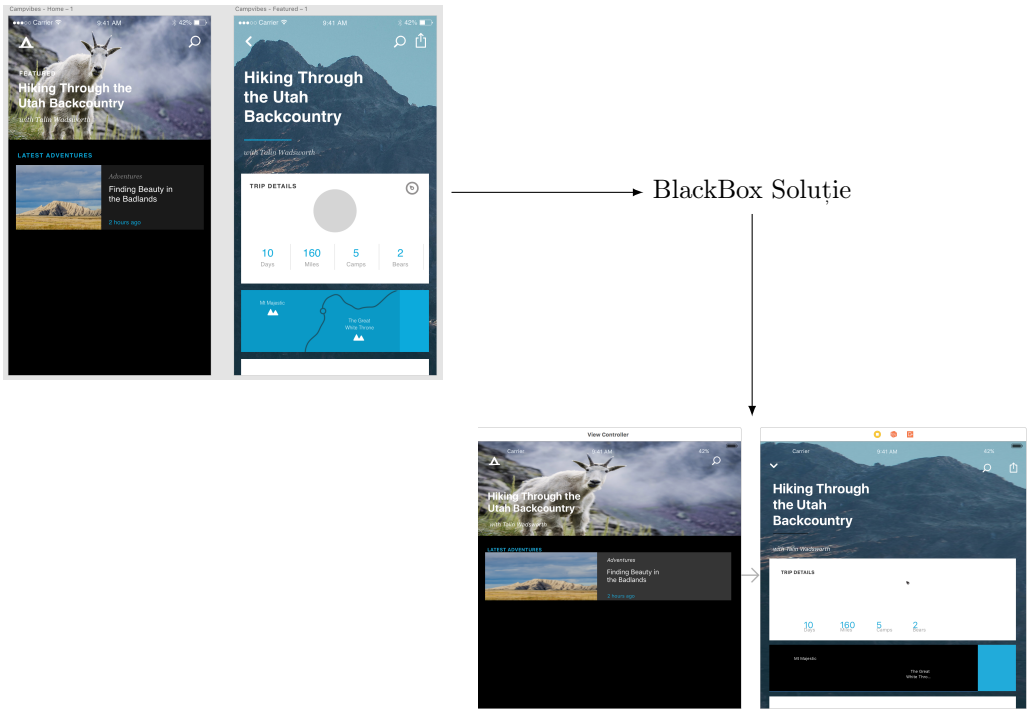


Figure 1.1: Prezentare Soluție

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, elmentele 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 salveaza 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 translatare generică între cele două reprezentări specifice XCode-ului si Adobe XD-ului: xml, respectiv json. Astfel, soluția propusă se poate extinde, putând fi folosită în diferite domenii de aplicații.

1.1.3 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.

Chapter 2

Prezentarea schemelor de reprezentare

Așa cum s-a precizat în secțiunile anterioare, proiectul reprezintă un plug-in adus aplicației Adobe Experience Design, pentru interacțiunea cu IDE-ul XCode.

În acest capitol se va descrie modul de lucru și reprezentările corespunzătoare celor două reprezentări XCode, respectiv XD.

2.0.1 Managementul elementelor vizuale in 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. Suita XCode include pe lângă documentația Apple de dezvoltare și o aplicație built-in - Interface Builder, o componentă principală in 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 fișier .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 trazițiilor dintre scene.

XCode permite o serie de controllere 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 controllere, va trebui extinsă schema de translatare care va fi prezentată ulterior, conform specificațiilor.

2.0.1.1 Anatomia 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 impreună 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 carcateristicile 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:

```
1  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2  <document type="com.apple.InterfaceBuilder3.CocoaTouch.Storyboard.
       XIB" version="3.0" toolsVersion="7706" systemVersion="14E46"
       targetRuntime="iOS.CocoaTouch" propertyAccessControl="none"
       useAutolayout="YES" useTraitCollections="YES">
3    <dependencies>
4      <plugIn identifier="com.apple.InterfaceBuilder.
           IBCocoaTouchPlugin" version="7703"/>
5    </dependencies>
6    <scenes/>
7  </document>
```

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

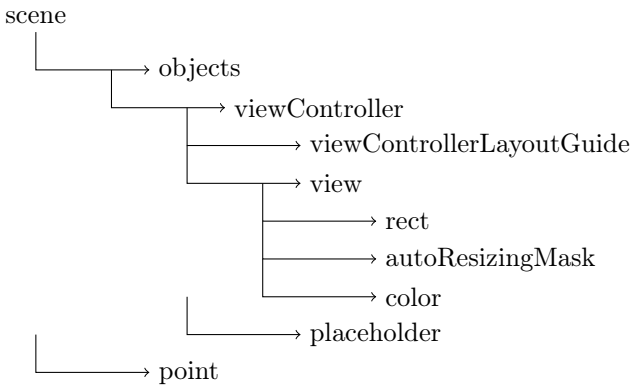
Tag-ul **dependencies** identifică orice dependență necesară storyboard-ului. Tag-ul copil **plugin** specifică pluginul necesar acestui storyboard.

Tag-ul **scenes** conține toate scenele (view controllers) din storyboard. Reprezentarea xml corespunzătoare este de forma:

```
1  <scene sceneID="TZ1-Tx-w3d">
2    <objects>
3      <viewController id="Z5v-SE-Hvj" sceneMemberID="viewController">
4        <layoutGuides>
5          <viewControllerLayoutGuide type="top" id="y7V-kb-M9q"/>
6          <viewControllerLayoutGuide type="bottom" id="GgS-WF-AQQ"/>
7        </layoutGuides>
8        <view key="view" contentMode="scaleToFill" id="0q9-ar-rJW">
9          <rect key="frame" x="0.0" y="0.0" width="600" height="600"/>
10         <autoresizingMask key="autoresizingMask" widthSizable="YES"
               heightSizable="YES"/>
11         <color key="backgroundColor" white="1" alpha="1" colorSpace=
               "calibratedWhite"/>
12       </view>
13     </viewController>
```

```
14      <placeholder placeholderIdentifier="IBFirstResponder" id="sPm-3H
            -Ntc" userLabel="My␣Responder" sceneMemberID="firstResponder"
            />
15    </objects>
16    <point key="canvasLocation" x="426" y="370"/>
17  </scene>
```

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



Un atribut important de menționat ar fi **id**, atribut prezent în aproape taote 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 noua 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 este de forma:

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

Fiecare View Controller conține o scenă principală și este reprezentată de tag-ul **view** .

```
1  <view key="view" contentMode="scaleToFill" id="0q9-ar-rJW">
2      <rect key="frame" x="0.0" y="0.0" width="600" height="600"/>
3      <autoresizingMask key="autoresizingMask" widthSizable="YES"
           heightSizable="YES"/>
4      <color key="backgroundColor" white="1" alpha="1" colorSpace="
           calibratedWhite"/>
5      <color key="tintColor" white="0.33333333333333331" alpha="1"
           colorSpace="calibratedWhite"/>
6  </view>
```

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

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

Figure 2.1 ,

reprezentarea xml asociată va fi de forma :

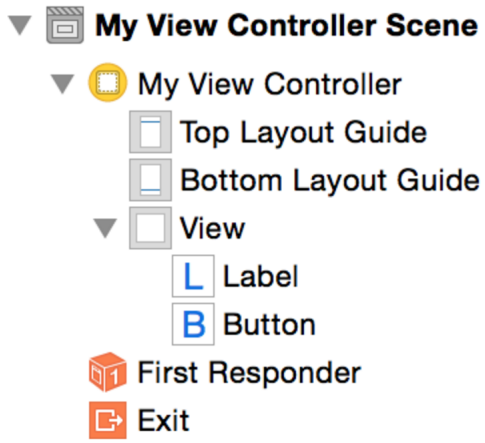


Figure 2.1: Views tag

```
<view key="view" contentMode="scaleToFill" id="0q9-ar-rJW">
   ...
    <subviews>
        <label opaque="NO" userInteractionEnabled="NO" contentMode="
           left" text="Label" lineBreakMode="tailTruncation"
           baselineAdjustment="alignBaselines" id="ngq-9d-cwY">
            <rect key="frame" x="279" y="129" width="42" height="21"
               />
            <color key="textColor" red="0.0" green="0.0" blue="0.0"
               alpha="1" colorSpace="calibratedRGB"/>
            <nil key="highlightedColor"/>
        </label>
        <button opaque="NO" contentMode="scaleToFill" fixedFrame="
           YES" ... id="mce-tZ-GZN">
            <rect key="frame" x="277" y="201" width="46" height="30"
               />
            ...
        </button>
    </subviews>
</view>
```

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 controllere 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 controllere sau butoane și alte view controllere.

```
 <viewController id="Z5v-SE-Hvj" customClass="MyViewController"
    sceneMemberID="viewController">
                    ...
                    <connections>
                        <segue destination="tr7-0U-dbu" kind="show"
                           identifier="
                           Segue_ShowSecondViewController" id="bCn-
```

```
                                    xA-8eX"/>
                            </connections>
                        </viewController>
}
}
```

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

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

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. Acesta a stat la baza protocoalelor 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.

2.0.2 Managementul elementelor vizuale in XCode

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.

2.0.2.1 Anatomia 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.

Directorul **artwork** conține o serie de fișiere cu informații în format json despre elementele vizuale ale fiecărei scene[1] în parte.

Directorul **resources** conține o serie de imagini folosite în designul XD, salvate local sub numele id-ului corespunzător. De asemenea, în cadrul acestui director se află informații legate de offeturile 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.

[1] Se vor folosi termenii scene și artboard interschimbabil

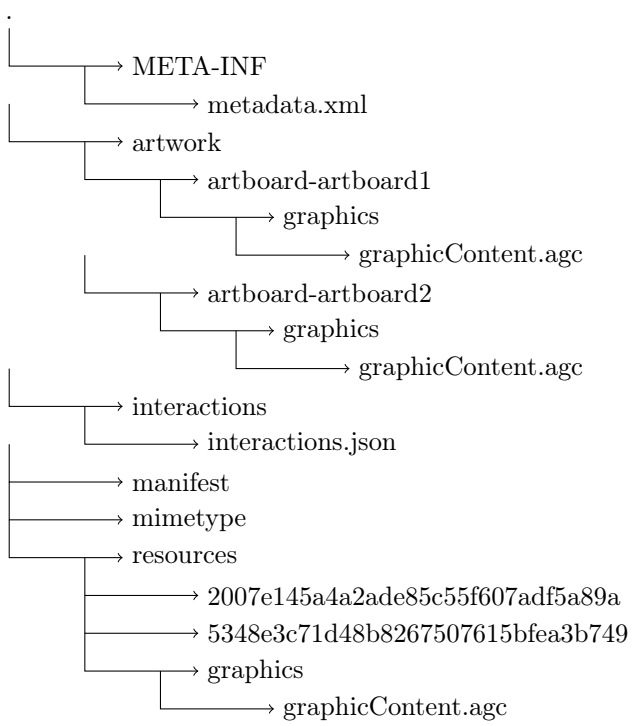


Figure 2.2: Ierarhie fișiere

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, **resoures**, respectiv **artboards** fac legătura cu restul fișierelor de descriere. Proprietatea **children** descrie lista de elemente vizuale ale scenei, precizând, astfel, pentru fiecare entitate, carcateristicile specifice, cadranul de încadrare și offsetul față de ierarhia de elemente față de care depinde.

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 asocieri între chei și valori. O pereche cheie-valoare din dicționar se numește un “entry". În interiorul dicționarului, cheile sunt stringuri unice, nenule. Tipul valorii nu are restricții, în afară de imposibilitatea declarării unei valori nule.

Chapter 3

Prezentarea schemelor de translatare

În acest capitol se va prezenta modul de translatare între cele două reprezentăriȘ 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 protocoale 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.

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

3.0.1 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 sa 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 ?

De exemplu, fie figurile **TODO:** 2 figuri care reprezintă un text în XCode, respectiv în XD. 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 notații care ne ajută în realizarea acestor asocieri.

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

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

Definiție 3.3. *Fie string-ul "x1.x2. ... .xn" care respectă definiția 3.3. Împărțim stringul inițial 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 definiția .*

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

```
{"textField" : {
        "type" : "text",
        "name" : "$text",
        "transform" : {
            "a" : "1",
            ...
            "tx" : "$rect.x",
            "ty" : "$rect.y"
        },
        "style" : {
            "fill" : {
                "type" : "solid",
                "color" : {
                    "mode" : "RGB",
                    "value" : {
                        "r" : "$color.red",
                        "g" : "$color.green",
                        "b" : "$color.blue"
}
```

Figure 3.1: Schemă translatare xcode->xd

Pentru a exemplifica definițiile de mai sus, ne vom folosi de figura 3.1 și de string-urile $color.red, $text.

Procedeul de aplicare al definițiilor este următorul:

- Pentru $color.red (cheia “r"):
  1. Observăm că stringul este o valoare temporară (care va trebui să fie înlocuită) conform definiției 3.1
  2. Evaluam valoarea stringului; Observăm că stringul este de tipul definiției 3.3
  3. Împărțim stringul color.red într-un vector de substringuri delimitate de “." => generalDict = [color, red] unde n = |generalDict| = 2
  4. Vectorul ne spune faptul că primele n-1 substringuri din generalDict vor fi chei, iar ultimul substring va reprezenta valoare (conform definiției 3.3); Așadar, $color.red va fi înlocuit cu valoarea găsită la atributul “red" al tagului “color".

Definiție 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 “.".*

Definiție 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 sa 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 3.2.

```
{"imageView."    :   "style.fill.pattern.href",
    "rect." : {
        "x" :   "textField.transform.tx",
        "y" :   "textField.transform.ty",
        "width" : {
            "rect"  :   "rect.shape.width",
            "text"  :   "textField.shape.width",
            "shape" :   "imageView.style.fill.pattern.width"

        },
        "height" : {
            "text"  :   "textField.shape.height",
            "rect"  :   "rect.shape.height",
            ...
}
```

Figure 3.2: Schemă translatare mapDict

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 parcurge fișierul storyboard asociat aplicației XCode prin intermediul clasei NSXMLParser. Pe măsură ce se parcurge fișierul, obținem numele tagului curent și un dicționar cu atributele și valorile asociate.

3. Verificăm dacă există vreo numele tagului curent apare 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 preluam 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 tags Se parcurge calea definită sub forma “c1.c2. ... .cn" (conform definiției 3.5) din dicționarul rootDict; Se obține o valoare Y sub forma “$x1.x2. ... .xn"; În continuare se procedează similar algorimtului din 3.0.1.
        - dicționar :

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

3.0.1.1 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.

3.0.1.2 Translatarea Interacțiunilor XCode->XD

O interacțiune în XCode, denumită “segue", este o conexiune care reprezintă tranziția 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 vede în figura 3.4. Soluția propusă are suport doar pentru Show segue.

Figure 3.4

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.

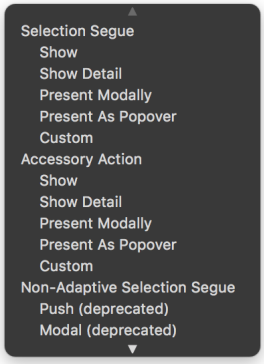


Figure 3.3: Switch tag

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

De asemenea, trebuie reținut ultimul id întalnit. Așa cum se vede în figura 2.0.1.1, 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.

3.0.2 Translatarea din Adobe Experience Design în XCode

TODO:
adobe -> text frame porneste din stanga jos; xcode -> text frame porneste din stanga sus

Exportul elementelor vizuale din Adobe XD în Xcode presupune parcurgerea fiserelor "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 explicate mai jos.

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

Definiție 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*

```
{...
"text"     :   "label",
"shape"     :   {
      "shape.type.rect" : {
          "style.fill.type.pattern" : "imageView",
          "style.fill.type.solid"   : "$frame"
      },
      "shape.type.path" : {
          "style.fill.type.solid" : "$imageView"
      },
      "shape.type.line" : {
          "style.fill.type.none" : "$lineView"
      }
...
```

Figure 3.4: XD2XCode Defs

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 definiției 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 carui 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 nivelelor 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 numită categorie de asseturi (de ex. pentru shape), pot exista mai multe modele de reprezentare, sau mai multe tipuri de asocieri, În funcție de fișierul .xml.*

În cazul în care tipul obținut în urma aplicării algoritmului menționat mai devreme, se obține un string de forma definiției 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 intr-un dictionar. Organizarea acestei scheme de translatare .json este realizată după modelul din Figura (2). Tag-urile din xml se pot împarți 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(2).

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 reprezentări se bazează pe "mostenire". Această relație garantează faptul că un tag care apare la un nivel inferior, va moșteni atributele părintelui, aka atributele copilului sunt "relative" la

```
{
"view" : {
        "header": {
            "key" : "view",
            ...
            "toString" : ["key", "contentMode", "id"] },
        "toString" : ["header", "rules"],
        "rules" : {
            "$header.id" : "$rand",
            ...
            "subviews" : {
                "$children.$sceneNo.artboard.children" : {}},
            "color" : {
                "$children.$sceneNo.style.fill.color.value" : {
                    "red" : "$r",
                    ...
                    "colorSpace" :"calibratedRGB" }},
            "toString" : ["rect", "autoresizingMask", "subviews", "
               color"],
            "order" : ["$header.id", "rect", "autoresizingMask", "
               color", "subviews"] }
        .... }
```

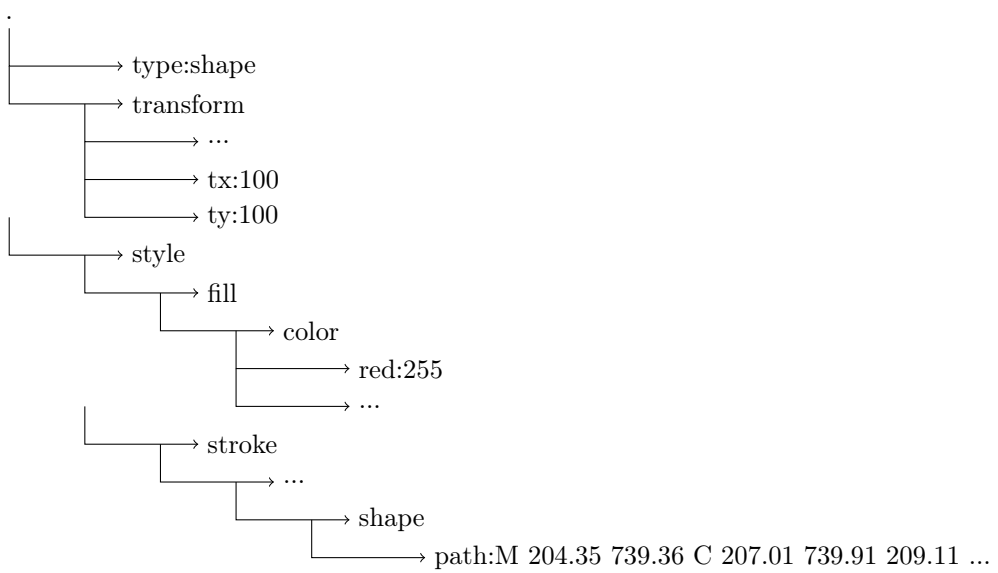
Figure 3.5: XD2XCode Schema

atributele părintelui. O bună exemplificare a acestei proprietăți este "mostenirea" 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 cadranul 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 reprezenat dependența tagului Key față de valorile "KeyDep". Maparile explicite 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". De asemenea, trebuie precizată structură de tip "<strîng>.<...>.<strîng>" care reprezintă nivelel ierarhice din cadrul dictionatului care trebuie să fie parcurse. Această reprezentare trebuie să asigure faptul că având o înșiruire de stringuri separate prin punct (n șiruri), n șiruri vor fi chei în dicționarul de translatare.

3.0.2.1 Translatarea Path-urilor și a diverselor figuri 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 manipulate cu precizie ridicată.

Un path este reprezentat în XD sub forma unei proprietăți json sub forma:



Se poate observa că 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 este de forma

```
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 ..." transform="
         translate(448 177)"/>
12 </svg>
```

Pentru translatarea din formatul json în formatul xml este nevoie de substituirea proprietăților **viewBox**, **fill**, **stroke**, **d** din **path** cu caracteristicile din json.

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

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

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

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

3.0.2.2 Translatarea grupurilor

Un grup în XD reprezintă o organizare logică a mai multor asset-uri într-un singur element vizual. 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 forma unui json al unui grup este specificată în figura 3.7.

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

Figure 3.6: Group

Proprietatea “transform" specifică modul de translatare sau rotire al 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 - view . Pentru construirea dimensiunilor view-ului, se va realiza o parcurgere inversă de la elementele constitunte grupului, la dimensiunea grupului. Așadar se iterează prin colecția de elemente din grup și se calculează offsetul minim și maxim al axelor x și y pentru fiecare. Cadranul grupului va fi încadrat între offsetul minim al axelor x și y și offsetul maxim al acestora.

3.0.2.3 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 decomprimare, așa cum s-a prezentat în figura 2.2 conține un director resources care, pe lângă fișierul de descreiere 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.

3.0.2.4 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 tranziții 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 sa 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 mapDict.

3.0.3 Sincronizarea scenelor între cele două reprezentări

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

3.0.4 Sincronizarea Adobe Experience Design - XCode

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 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 refăcut.

3.0.5 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.

3.0.6 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, va rezulta 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 3.8, iar al exportului, prin intermediul Clipboardului, se regăsește în figura 3.9.

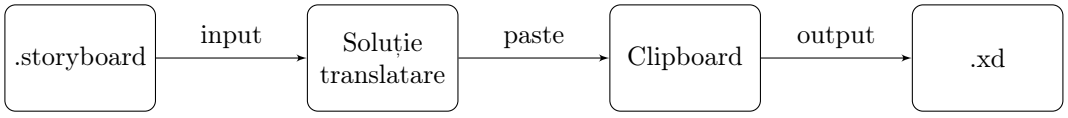


Figure 3.7: Clipboard Import

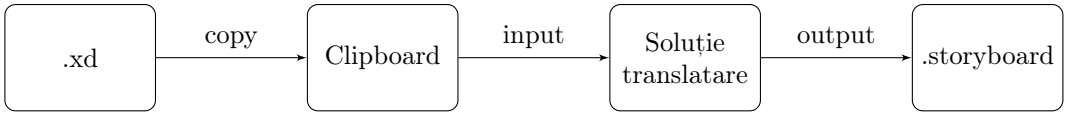


Figure 3.8: Clipboard Export

În a doua etapă - etapa finala - s-a renunțat la folosirea Clipboardului, în favoarea creării fișierului xd. Acest lucru s-a realizat din două motive: În primul rând, folosirea Clipboardului implica 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 tranzițiilor prin Clipboard.

Așadar, pentru preluarea informațiilor din cadrul unui fișier xd, se va realiza decomprimarea acestuia prin apelarea comenzii **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 fi apoi comprimată în format .xd. Această comprimare se va realiza prin intermediul comenzii **zip -r –no-dir-entries <ierarhie fișiere> <design xd>**.

Arhitectura acestui model de lucru se regăsește în figurile 3.11, 3.10.

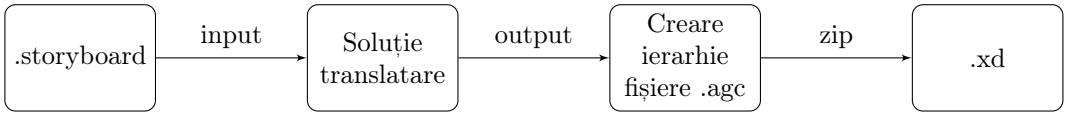


Figure 3.9: Simple Import

Figure 3.10: Simple Export

Soluția prezentată (sau BlackBox-ul prezentat în figura 1.1) se împarte într-o serie de module care implementează diferite concepte și funcționalități:

- 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)

Aceste module și interacțiunea dintre ele este reprezentată în figura 3.12.

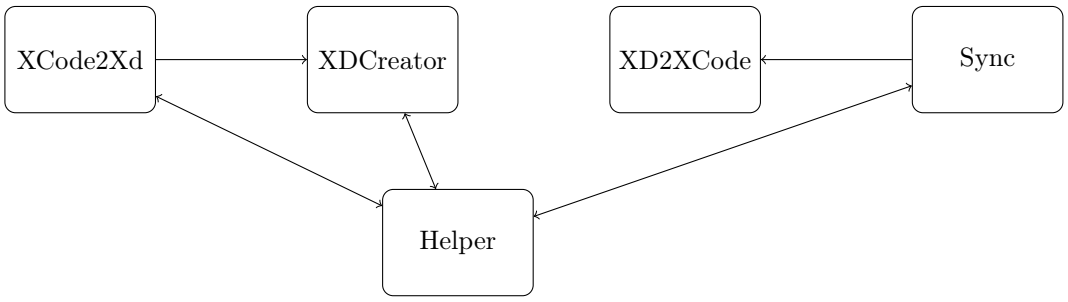


Figure 3.11: Arhitectură

3.0.7 Performanță

Realizarea translatării 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 -)

3.0.8 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 controllere (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 dependențeș
2. Translatarea multiplelor path-uri/figuri geometrice care aparțin aceluiași grup într-o singură imagine
3. Reprezentarea vizuală a mapărilor - pentru extinderea facilă a aplicației

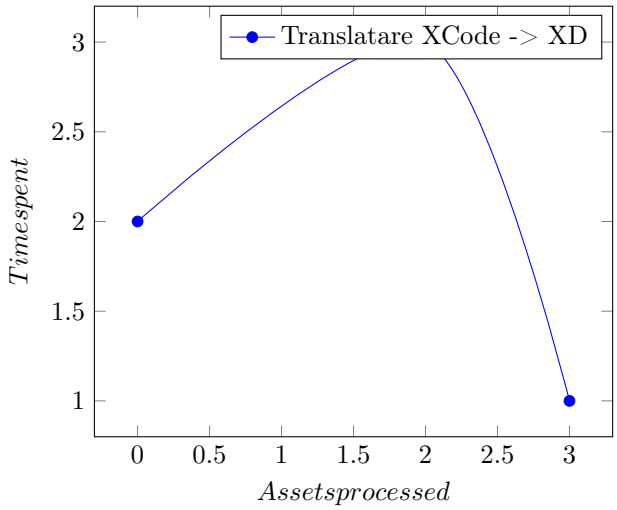


Figure 3.12: Grafic XCode XD

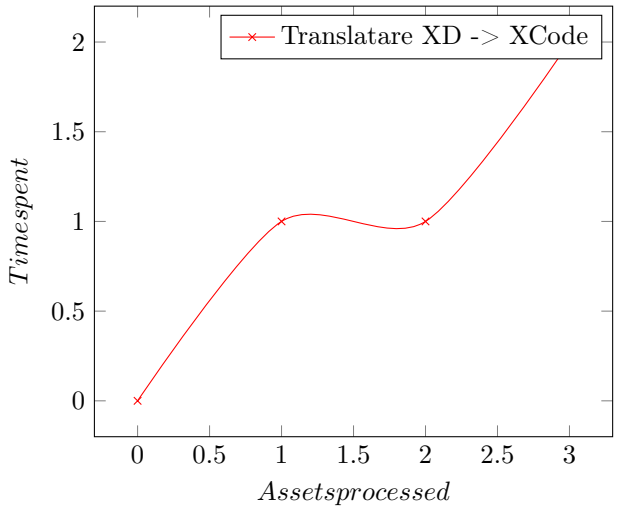


Figure 3.13: Grafic XD XCode

3.0.9 Concluzii

În cadrul acestui proiect am prezentat o schemă de translatare între doua 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.

Această schemă de translatare se poate folosi în primul rând pentru managementul elementelor vizuale, așa cum este prezentă soluția de fața, dar se poate extinde pentru o serie de aplicații.

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)

Appendix A

Project Build System Makefiles

A.1 Makefile.test

```
1  # Makefile containing targets specific to testing
2
3  TEST_CASE_SPEC_FILE=full_test_spec.odt
4  API_COVERAGE_FILE=api_coverage.csv
5  REQUIREMENTS_COVERAGE_FILE=requirements_coverage.csv
6  TEST_REPORT_FILE=test_report.odt
7
8
9  # Test Case Specification targets
10
11 .PHONY: full_spec
12 full_spec: $(TEST_CASE_SPEC_FILE)
13         @echo
14         @echo "Generated␣full␣Test␣Case␣Specification␣into␣\"$^\""
15         @echo "Please␣remove␣manually␣the␣generated␣file."
16
17 .PHONY: $(TEST_CASE_SPEC_FILE)
18 $(TEST_CASE_SPEC_FILE):
19         $(TEST_ROOT)/common/tools/generate_all_spec.py --format=odt
               -o $@ $(TEST_ROOT)/functional-tests $(TEST_ROOT)/
               performance-tests $(TEST_ROOT)/robustness-tests
20 #
21
22 # ...
```

Listing A.1: Testing Targets Makefile (Makefile.test)