스마트폰 앱의 고급화를 위한 동적화면 저작도구 시스템의 설계 및 구현

김병수*, 강미애**, 이봉림***
(주)야긴스텍 기술연구소
e-mail:(bskim, makang, lbr86)@yagins.com

Design and Implementation of Dynamic Screen Authoring Tool For Advanced Smartphone App

Byung-Soo Kim*, Mi-Ae Kang**, Bong-Rim Lee***
R&D Center, YAGINSTEK Co.,Ltd

요 약

최근 스마트폰 앱의 공급이 급속히 늘어남에 따라 사용자의 눈높이는 갈수록 높아지고 있다. 이에 따라 스마트폰 앱 공급자는 앱을 제작할 때 좀 더 유익한 기능을 제공하기 위하여 노력해야 함은 물론이고 사용자에게 좀 더 흥미를 유도할 수 있는 시각적인 요소를 부각시키기 위하여 많은 노력을 기울여야 한다. 그런 시각적인 요소를 프로그램 개발자가 코드 구현만으로 표현하기에는 많은 어려움이 따르며 개발 기간이 그만큼 늘어나게 된다. 본 논문에서는 시각적인 측면에서 좀 더 고급화된 스마트폰 앱을 제작하기 위하여 역동적이고 차별화된 앱을 제작할 수 있는 동적 화면 저작도구를 제안한다.

1. 서론

스마트폰 앱의 공급이 늘어남에 따라 사용자의 눈높이는 갈수록 높아지고 있다. 이에 따라 스마트폰 앱 공급자는 앱을 제작할 때 기능적 측면과 시각적 측면을 모두 고려해야 한다. 아무리 기능이 많은 앱이라도 시각적인 표현이 부족하다면 사용자에게 흥미를 유도하지 못할 수 있고 반대로 기능이 비교적 적은 앱이라도 시각적인 표현이 풍부하다면 많은 흥미를 유도하여 관심을 더 끌 수도 있다.

본 논문에서는 시각적인 측면에서 좀 더 고급화된 스마트폰 앱을 제작할 수 있는 GUI환경의 동적 화면 저작도구를 제안한다. 기존 저작도구는 정적인 화면을 제작할 수 있는데 반해 제안하는 저작도구는 동적인 화면 제작이 가능하여 좀 더 역동적이고 차별화된 앱을 제작할 수가 있다. 개발자가 코드 구현만으로 역동적인 화면을 제작하기엔 많은 어려움과 시간이 소요되지만 제안하는 저작도구를 이용하게 되면 개발기간과 유지보수 기간을 단축시킬 수 있고 기간 대비 많은 시각적인 요소를 한층 더 고급화하여 표현할 수 있게 된다. 또한 프로그램 개발자가 아니어도 누구나 저작도구를 이용하여 화면을 제작할 수 있기 때문에 기능 구현 부분과 UI 구현 부분을 분담하여 작업할 수 있으며 기획자 또는 디자이너의 표현 의도가 최대한 정확하게 반영될 수가 있다.

본 논문에서 2장은 기존 저작도구에 대하여 살펴보고 3장에서는 제안하는 저작도구의 설계 및 구현 내용에 대하여 설명하고 4장에서는 실험 및 결론으로 마무리 한다.

2. 관련연구

스마트폰 앱을 제작하기 위한 저작도구로 이미 잘 알려진 m-Bizmaker 와 MIT App Inventer 저작도구를 분석해 보았다.

2.1. m-Bizmaker

m-Bizmaker는 스마트 디바이스에서 운영할 비즈니스용 앱 프로그램을 쉽고 빠르게 개발하는 저작도구이다. GUI방식의 도구를 활용하여 누구나 쉽게 사용할 수 있는 도구이며 사용자 인터페이스(UI)의 구현부터 비즈니스 로직 구현, DB테이블 생성까지 자동화 해 주는 저작도구이다.[1]

프로그램개발자가 아니어도 일반인이 스마트폰 앱을 제작할 수 있는 장점이 있지만 동적인 요소를 제작하는 기능은 제공하지 않는다.

2.2 MIT App Inventer

MIT App Inventer는 프로그래밍을 시작하는 초보자거나 안드로이드 프로그래밍이 능숙한 개발자 모두가 사용할 수 있는 블록기반의 저작도구이다. 코드 구현 방식보다 훨씬 적은 시간에 더 많은 복잡한 앱을 만들 수 있도록 해주는 도구이다.[2]

UI가 직관적인 장점이 있지만 동적인 요소를 제작하는 기능은 제공하지 않으며 Android 기반의 앱만 생성이 가능하다.

3. 저작도구의 설계 및 구현

이번 장에서는 본 논문에서 제안하는 저작도구의 시스템 구성도, 저작도구의 동적화면 구성원리와 기능, 실제 디바이스에서 구동시키기 위한 앱 실행 데이터 및 앱 실행 엔진에 대하여 설명한다.

3.1 시스템 구성도

저작도구 시스템은 [그림1] 과 같이 크게 화면을 제작할 수 있는 도구 영역과 실제 디바이스에서 실행시키기 위한 기능을 수행하는 앱 실행 라이브러리 영역으로 나뉜다.

[그림 1] 시스템 구성도

저작도구를 이용하여 화면을 제작하고 실제 디바이스에서 화면을 구동시키기 위한 절차는 아래와 같다.

(1) 화면 제작자는 저작도구를 이용하여 동적인 화면을 제작한다.

(2) 저작도구의 메뉴를 통해 앱 실행 데이터를 생성한다.

(3) 프로그램 개발자는 개발 프로젝트에서 앱 실행 라이브러리를 참조 추가한다.

(4) 저작도구에서 생성한 앱 실행 데이터를 개발 프로젝트에 리소스 형태로 추가시킨다.

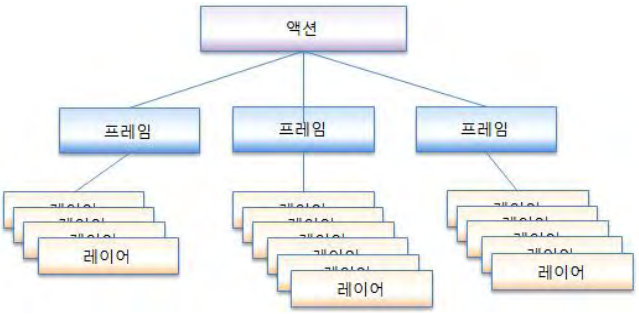
(5) 앱 실행 라이브러리가 제공해 주는 API(Application Programming Interface)를 통해 원하는 화면을 실행시킨다.

(1), (2)번의 저작도구에서의 제작 과정은 프로그램 개발자가 아닌 개발 경험이 없는 기획자 또는 디자이너가 직접 작업에 참여할 수 있다.

3.2 저작도구에서의 동적 화면 구성원리

저작도구에서 사용자가 제작하는 동적인 화면의 구성원리는 다음과 같다. 먼저 저작도구에서 미리 등록한 이미지를 화면에 배치하여 정적인 화면을 제작하게 되고 제작한 여러 정적인 화면을 순서대로 재생하게 되면 하나의 동적인 화면이 완성되어 진다. 화면에서 애니메이션 되는 이미지들의 움직임 표현을 GUI 환경의 저작도구에서 마우스 움직임만으로 표현할 수 있어서 코든 구현 방법보다 훨씬 짧은 시간 내에 원하는 애니메이션 즉 동적인 화면을 제작할 수가 있게 된다.

시스템 내부적으로 하나의 동적인 화면은 액션 객체에 저장이 되며 정적인 화면은 프레임 객체에 저장이 되고, 정적인 화면에 배치된 이미지들은 레이어 객체에 저장이 된다. 각 객체들 간에는 상, 하위 계층 구조를 가지며 [그림2]는 객체간의 계층 구조를 그림으로 나타낸 것이다. 여러 레이어 객체들이 모여 하나의 프레임 객체가 되며 여러 프레임 객체가 모여 하나의 액션 객체가 된다.



[그림 2] 동적 화면을 구성하는 객체 간의 계층구조

3.3 저작도구의 이미지 편집기능

모바일 환경에서는 앱 용량을 최소한으로 줄이기 위한 노력이 필요하다. 화면에 배치된 이미지 즉 레이어 객체에 여러 가지 이미지 효과를 적용하여 활용하면 하나의 이미지로 여러 이미지를 추가한 효과를 낼 수가 있다. 결국 이미지 추가를 최소화 하여 앱 용량을 줄이는 데 기여할 수가 있다.

본 논문에서 제안하는 저작도구에서 제공하는 이미지 편집기능으로는 이미지의 확대/축소, 회전/반전, Color Filter, Blending 효과, Blur 효과 적용 등이 있다. 원본 이미지의 비트맵 정보를 추출하여 Alpha, Red, Blue, Green 값을 적용하고자 하는 효과에 맞는 알고리즘을 적용하여 변경하여 비트맵을 재구성하는 원리이다. [그림3]은 이미지 제어기능 중에 Color Filter 효과를 적용한 예시를 보여주는 그림이다. 왼쪽부터 원본 이미지, Gray Filer 적용 이미지, Sepia Filter 적용 이미지 순으로 보여주고 있다.

3.4 저작도구의 시뮬레이션

본 논문에서 제안하는 저작도구에서는 사용자가 제작

[그림 3] 이미지 효과적용 예제

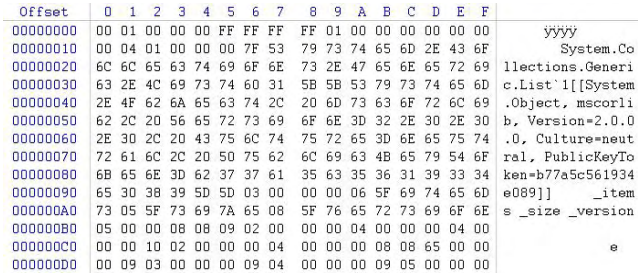
한 동적인 화면 결과물을 저작도구 상에서 바로 확인할 수 있는 시뮬레이션 재생 기능을 제공한다. 재생 버튼을 누르게 되면 내부적으로 타이머 루프가 동작하여 정적인 화면 단위인 프레임들을 순차적으로 화면에 그려주어 동적인 화면이 재생되는 원리이며 위지윅(WYSIWYG) 방식으로 실제 디바이스에서 구동되는 화면과 동일하다.

3.5 앱 실행 데이터

저작도구에서 제작한 결과물을 실제 디바이스 화면에서 실행하기 위해서는 앱 실행 데이터가 필요하다. 앱 실행 데이터는 액션 객체가 포함하고 있는 프레임 객체 정보, 레이어 객체 정보와 화면 재생에 필요한 그 외 모든 정보들이 바이트 배열로 조합되어진 데이터이다. 바이트 형태로 이루어진 바이너리 데이터는 모든 스마트폰 OS 에서 사용할 수 있다. 또한 불필요한 공간을 최대한 줄이기 위하여 데이터 크기에 최적화 된 자료형(2byte 이하)을 사용하여 XML 형태로 생성되는 데이터[3,4] 보다 적은 용량을 차지하게 된다. [표1] 은 바이트 배열로 조합할 때의 데이터 구조에 대한 일부 예시를 나타낸 것이며 [그림 4]는 생성한 앱 실행 데이터를 바이너리 파일 에디터로 읽었을 때의 예시화면이다.

[표1] 앱 실행 데이터 구조

필드	단위(크기)	설명
Action Count	short(4byte)	총 액션 개수
Action	byte	액션 index
Included FrameCount	short(4byte)	액션이 가지고 있는 프레임 개수
Frame Offset	short(4byte)	해당 액션의 프레임offset
FrameCount	short(4byte)	총 프레임 개수
Included LayerCount	short(4byte)	프레임이 가지고 있는 레이어 개수
LayerOffset	short(4byte)	해당 프레임의 레이어offset
LayerCount	short(4byte)	총 레이어 개수
imgNo	short(2byte)	레이어의 이미지 No
x	short(2byte)	레이어의 x 좌표
y	short(2byte)	레이어의 y 좌표
property	short(2byte)	레이어의 변경된 속성 정보



Offset	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
00000000	00	01	00	00	00	FF	FF	FF	FF	01	00	00	00	00	00	00	ÿÿÿÿ
00000010	00	04	01	00	00	00	7F	53	79	73	74	65	6D	2E	43	6F	System.Co
00000020	6C	6C	65	63	74	69	6F	6E	73	2E	47	65	6E	65	72	69	llections.Generi
00000030	63	2E	4C	69	73	74	60	31	5B	5B	53	79	73	74	65	6D	c.List`1[[System
00000040	2E	4F	62	6A	65	63	74	2C	20	6D	73	63	6F	72	6C	69	.Object, mscorli
00000050	62	2C	20	56	65	72	73	69	6F	6E	3D	32	2E	30	2E	30	b, Version=2.0.0
00000060	2E	30	2C	20	43	75	6C	74	75	72	65	3D	6E	65	75	74	.0, Culture=neut
00000070	72	61	6C	2C	20	50	75	62	6C	69	63	4B	65	79	54	6F	ral, PublicKeyTo
00000080	6B	65	6E	3D	62	37	37	61	35	63	35	36	31	39	33	34	ken=b77a5c561934
00000090	65	30	38	39	5D	5D	03	00	00	00	06	5F	69	74	65	6D	e089]] _item
000000A0	73	05	5F	73	69	7A	65	08	5F	76	65	72	73	69	6F	6E	s _size _version
000000B0	05	00	00	08	08	09	02	00	00	00	04	00	00	00	04	00	
000000C0	00	00	10	02	00	00	00	04	00	00	00	08	08	65	00	00	e
000000D0	00	09	03	00	00	00	09	04	00	00	00	09	05	00	00	00	

[그림 4] 에디터 상에서 앱 실행 데이터 로드 화면

3.6 앱 실행 라이브러리

저작도구에서 제작한 결과물을 실제 디바이스 화면에서 실행시키는 역할을 하는 라이브러리이며, 스마트폰 OS 별로 라이브러리가 제공되어야 한다. 앱 실행 라이브러리는 크게 저작도구에서 생성한 앱 실행 데이터를 설계된 데이터 프로토콜 구조를 참조하여 해석하는 부분과 저작도구에서 제작한 결과물을 디바이스 화면에 표현하는 부분으로 나뉜다. 앱 실행 라이브러리는 프로그램 개발자 프로젝트에 참조되어 사용되어 지며 프로그램 개발자는 제공해주는 라이브러리 API 문서를 참고하여 원하는 결과물을 디바이스 화면에 표현하게 된다. [그림5]는 프로그램 개발자가 앱 실행 라이브러리 API를 사용하여 100번 ID를 가지고 있는 액션을 화면에 그리기 위한 코드 구현 예시이다. 몇 줄의 코드 구현만으로 원하는 결과물을 디바이스 화면에 표현할 수 있게 된다.

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view from its nib.

    //init RuntimeLibrary
    RuntimeManager *runtimeManager = [[RuntimeManager alloc]init];

    //create view (X:0, Y:0, Width:640, Height:960)
    [runtimeManager createView_X:0 Y:0 Width:640 Height:960];

    //create Action (id:100)
    ActionObject *action = [runtimeManager createActionObject:100];

    //draw Action
    [runtimeManager drawAction:action];

}
```

[그림 5] 앱 실행 라이브러리 사용 예시

3.7 저작도구 화면 레이아웃

[그림6]은 저작도구의 화면 레이아웃으로 크게 객체관리뷰, 프레임뷰, 이미지관리뷰, 작업뷰, 레이어뷰, 툴바메뉴로 구성되어져 있다.

객체관리뷰는 사용자가 생성한 모든 객체 그룹을 리스트 형태로 보여주고 프레임뷰는 객체 하위에 포함된 프레임들을 보여주며 레이어뷰는 프레임 하위에 포함된 레이어들을 보여준다. 작업뷰는 저작도구의 실제 작업 화면이 되며 사용자가 현재 선택한 프레임의 화면을 보여준다. 실제 저작도구의 핵심인 화면 제작 기능을 수행하기 위한 영역이며, 사용자는 이미지관리뷰에서 미리 등록한 이미지 오브젝트를 선택하고 작업뷰의 원하는 위치에 배치함으로