ISSN 2395-1621

Visual Mobile AppInventor for Mobile End-Users

#1Kalyani K. Devnikar, #2Dr. S.B. Sonkamble

1devnikarkalyani15@gmail.com
2sonkamblesulochana@gmail.com

#1Department of Computer Engineering,
#2Prof. Department of Computer Engineering,

JSPM's Rajarshi Shahu School of Engineering & Research, Narhe,
Pune, Maharashtra, India.

ABSTRACT

Mobile phone applications are becoming more and more popular. As the technology advances, services like camera, accelerometers, compass, GPS, etc. are provided by the mobile phones. However, the more smart phones become smarter, the more the design of complex applications covering the various user needs becomes a complicated task. To support an End-User in the development of mobile applications we present a visual approach to enable End-Users to compose visually their own applications directly on their mobile phone. The methodology enables the end-user to develop applications and/or compose services on the smart phone, so making the way towards new scenarios where smart phones replace and overtake the Personal Computer, given their native possibility of wide connectivity, when augmented by features for interaction with remote systems and sensors. Moreover, the results of a preliminary usability study revealed a good satisfaction degree and the effectiveness and efficacy of the MicroApp visual approach.

Keywords: Visual languages; Service Composition; Mobile End-User Development; Mobile Applications

ARTICLE INFO

Article History

Received: 28th June 2016

Received in revised form :

28th June 2016

Accepted: 30th June 2016

Published online :

1st July 2016

I. INTRODUCTION

The term visual computing is concerned with the images and 3D models, i.e. computer graphics, image processing, visualization etc. In this paper, we present a visual approach to enable End-Users to compose visually their own applications directly on their mobile phone. The definition of an approach supporting an End-User in the development of mobile applications is a hard task because of the characteristics and the limitations of mobile device interfaces. The more smart phones become smarter, the more the design of complex applications covering the various user needs becomes a not easy task. The number of available applications for smart phones is rapidly growing, together with the number of users interested in top-range mobile devices. An appropriate handling of mobile services allows the user to define by himself more complex applications meeting different needs. The composition of these applications can be visually modeled through graphical symbols, associated to a particular application behavior and to a specific user interface. By opportunely connecting these graphical symbols, the user can describe complex behaviors Recent advances in mobile technology, mobile Networks and mobile computing offer new functionalities and applications for software systems on mobile devices. The demand for mobile applications comes from a wide range of domains. New services and innovative interaction modalities are continuously proposed, including gesture detection, device movement and context-based control. Smart phones should allow the user to combine services across multiple instruments to get smarter applications. To this aim, we propose a touchable interface and an ad-hoc visual language, enabling the user to compose simple focused applications, named MicroApps. The MicroApp Generator tool improves the effectiveness in terms of time and editing errors with respect to the use of MIT App Inventor. This paper is structured as follows. Related work is discussed in Section II and Section III describes system in detail. Section IV discusses designing of a system and section V gives testing method and section VI is summary and conclusion.

II. RELATED WORKS

This application uses a visual programming language supporting all the programming constructs and a server to store projects and generate .apk files. However, there are stand-alone versions that run entirely on the PC. Cabana is a Web-based application supporting the development of multiple mobile platforms. Programming is based on a wiring diagram supplemented by the use of JavaScript. It is addressed to beginner computer science students.

a) Jigsaw Approach

Danado and Patern`o [1][2] adopt a jigsaw approach
to compose pervasive/Web service. The tool Puzzle needs the support of an external server to manage external objects and the application repository. It also provides an authoring tool for designing the User Interface, an HTML viewer and native modules to exploit device functionalities. Puzzle does not support static parameter definition, thus, it does not exploit the advantage of having a predefined application, with some information bounded at design time. Their approach gets the list of applications from an external Application Repository, without contingency management.

b) Microsoft TouchDevelop

Microsoft TouchDevelop [3] is a programming environment running on smart phones. The user writes scripts by tapping on the screen. It has built-in primitives which make it easy to access the sensor data available on a mobile device. It supports the combinations of phone sensor data (e.g., location) and the cloud (via services, storage, computing, and social networks). Differently from MicroApp Generator, the language is not graphical: it uses variables and assignment statements. The smart phone needs to be connected to the TouchDevelop server in order to generate an app **.**

c) Microservices

In [4] a mobile tool has been proposed to compose Microservices. They can be created considering two user expertise levels: beginner, enabling template based development of a Microservice, and advanced, based on an XML-based language. Experienced users may use a visual editor for editing the XM describing Microservices' profile, content, logic, and presentation[5].

d) Husky Tool

The HUSKY tool [6] enables the PC-users to compose the logic of a PC-application by spatially arranging the component services within spreadsheet cells, following the idea presented in[7]. The execution proceeds on the spreadsheet from left to right and from top to bottom. A set of adjacent cells makes a sequence of events. The information regarding the flow of data among cells is not graphically represented.

e) Mashup Tool

Marmite [8] is an end-user mashup composition tool. The system runs on the PC. It offers a set of operators such as Search, Extract and Filter by, to extract and process data from Web pages and Web services. A data-flow approach is adopted to chain operators. The flow of data is displayed in a table, adopting a spreadsheet view.

III VISUAL MOBILE APPROACH

The proposed approach helps the users to manage the complexity of their activities performed with the mobile device by composing simple applications. The users do not concentrate in managing the dataflow and in the designing of the user interface, but only on the sequence of the actions needed to model the required MicroApp.

In this proposed approach, we present a MicroApp Generator Architecture which gives us the basic idea of a structure of a system.

Microapp Generator Architecture:

The system architecture which introduce framework of MicroApp Generator.

MicroApp is registered in the action list launched when the selected activation event happens. Before executing a MicroApp, the activity Context Detection checks the user context. Then, the activity Contingency Management verifies

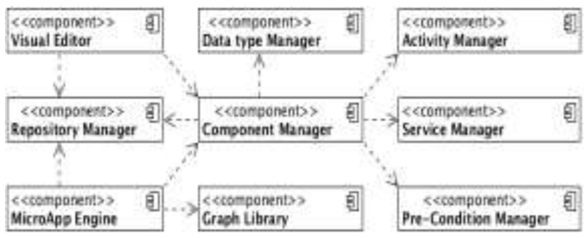

<<component>> Visual Editor
<<component>> Data type Manager
<<component>> Activity Manager
<<component>> Repository Manager
<<component>> Component Manager
<<component>> Service Manager
<<component>> MicroApp Engine
<<component>> Graph Library
<<component>> Pre-Condition Manager

Fig 1: MicroApp Generator Architecture

the availability of the involved services and tries to replace unavailable services.

In this way, the application is able to manage unpredictable availability of the involved services, i.e., faults or network connectivity problems. When all the required services are available, this activity provides as output a Service Graph, that represents the MicroApp design instantiated with the available services. The Execution activity linearizes the graph through a topological sort and starts the execution. The Execution activity periodically verifies if some context changes occur.

MicroApp Enactment. Before executing a MicroApp, the activity Context Detection checks the user context. Then, the activity Contingency Management verifies the availability of the involved services and tries to replace unavailable services.

During the MicroApp Modeling activity the user can try his MicroApp at any time, by enacting it. In this way the development process is incremental and allows experimentation and testing of partially complete applications. This feature lets end-user skills grow gradually and provides immediate satisfaction.

Visual Language:

Visual programming language (**VPL**) is any programming language that lets users create programs by manipulating program elements graphically rather than by specifying them textually. A VPL allows programming with visual expressions, spatial arrangements of text and graphic symbols, used either as elements of syntax or secondary notation. For example, many VPLs (known as *dataflow* or *diagrammatic programming*) are based on the idea of "boxes and arrows", where boxes or other screen objects are treated as entities, connected by arrows, lines or arcs which represent relations.

An ad-hoc developed mobile Visual Editor, designed considering the limited size of the device screen, supports the user in the modeling of the behavior of a MicroApp by composing its application logic.

The Visual Editor assists the selection of an action by highlighting the action icons whose input is compatible with the inputs/outputs of the blocks already positioned. Once selected the action, the user clicks on a specific column of the Composition Area to add the action. The editor still assists the user disabling the columns that are not compatible with the action to be inserted.

Mathematical Model:

In this section, mathematical model is represented in the form of set theory.

Let us consider a set S

Where, S= {U, R, SER, D,I,Ser,C}
Here, S: System which includes:
U: Set of Users Where U= {U1, U2, U3 …, Un}
SER: Server. R: Set of Request.
Where R= {R1, R2, R3……., Rn}
D: Database. N: Number of Cluster. (i.e. 2)
I:- Input From Users
Ser :- Services From the User
C:- Connection that services to the User Input

Let Us F final System

F=(S,O)
S:- System
O:- Output
O= { (I U Ser) * (R U D) / C }

IV. DESIGNING A APPINVENTOR

In this section we present a designing of a AppInventor.

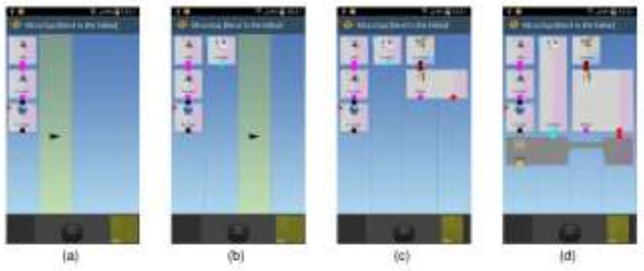


Fig. 2 designing steps of “send to editor”

Let us start by showing a sample application developed using AppInventor. Consider the following scenario: Marc is a reporter of the newspaper Daily News in a war zone. Repeatedly, he sends to his editor by email encrypted pictures labelled by the name of the place where each picture has been taken.

The steps Marc performs to design the application ”Send to the Editor” are depicted in Fig. 2, where services are represented by rounded rectangles and are connected through bullets which correspond to the service input/output parameters.

Services are classified by type of action or device sensor (e.g., Camera, Send, Facebook, Position) in the service catalog. To compose the application Marc performs the following actions: to take a picture, he selects the folder Camera, which collects all the services related to the device camera (*Camera.Take, Camera.Preview, Camera.Save*). By a first touch, he selects the service *Camera.Take* and, by a second touch, puts it in the first column (Fig. 2(a)). As he wants to see a preview of the picture before deciding to send it to his editor, he selects the service *Camera.Preview*.

The output bullet of *Camera.Take* is compatible with the input bullet of *Camera.Preview* since both have the same color (pink), denoting the image data type. Then, to encrypt the picture, he first discovers the service Encrypt available on the Web, and then puts it in the first column, since the output bullet of *Camera.Preview* is compatible with the generic input (black) of the selected service.

The editor’s contact is selected by adding the service *Contact.Static* in the first available column on the right as shown in Fig. 2(b). Marc also has to send the picture location information. Thus, as shown in Fig. 2(c), he selects the service Location, which detects Marc’s position (by means of the GPS of the smart phone), and puts this service in the third column.
Next, he connects this service to the service Maps, which determines the name of the user location and produces a map of his position.

With reference to Fig. 2(d), the service *Mail.Send* needs to collect the recipient email address, the picture and the location information. First Marc drags and drops the service
Mail.Send in the first empty space of the first column

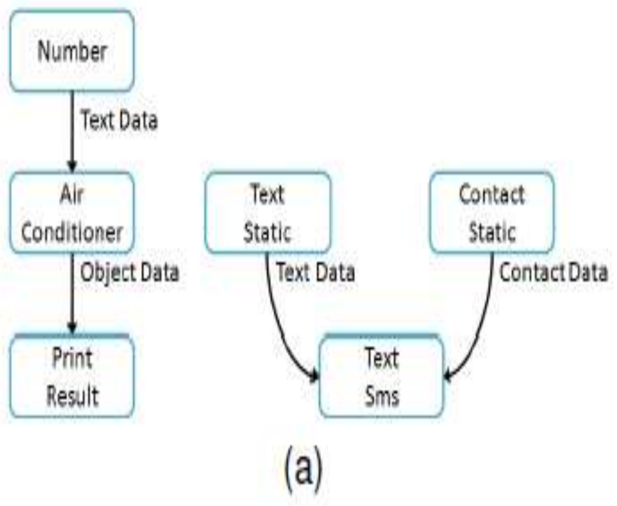
attaching it to Encrypt. Successively he touches the services Contact.Select and Maps to associate their output parameters to the inputs of the service *Mail.Send*. Moreover, possible coupling ambiguities are solved by prompting a popup menu to the user.

For example, *Mail.Send* has two parameters of type text (i.e., body and subject) both compatible with the location information outputted by Maps: from a popup menu, the user chooses to attach the location information to the body parameter. If there is an empty space between two services to be connected, the service icon is automatically lengthened, as in the case of *Contacts.Select* in the second column. Marc selects the editor's contact by long pressing on the service *Contact.Static*. A contact specified at design time (static) is set for all the successive uses of the application. He can specify more than one contact and the system will manage the collection of contacts in a transparent way. Similarly, Marc sets the password of the encryption algorithm for the service Encrypt at design time. In coupling parameters, the editor may automatically permute the input parameters in order to connect them correctly.

V. TESTING AND DEPLOYING A APPINVENTOR

Since we are using MicroApp Architecture for our proposed system, entire working of the newly developed application will completely follow the MicroApp execution procedure.

The data-flow of a MicroApp is represented by a directed acyclic graph since the MicroApp design does not use loops. Each service has a set of inputs generated by other services and, in turn, it provides inputs for other services. The service execution plan is automatically generated by a topological sort of the data-flow graph. The MicroApp Engine loads the XML description and translates it into a linear execution sequence by instantiating the service objects and running the process.



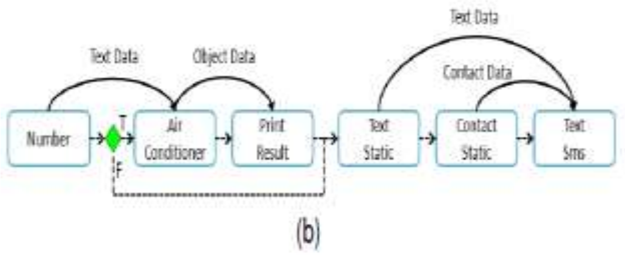


Fig. 3 The directed acyclic graph of a MicroApp (a) and its linearized execution plan (b).

As an example, Fig. 3(a) shows the directed acyclic graph. Figure 3(b) shows the linearized execution plan of the application. The solid arrows represent the data-flow, whilst the dotted lines describe the control flow. The pre-condition defined on the service AirConditioner is implemented as a decision node on the control flow. If the precondition is satisfied the service AirConditioner is executed, otherwise the control passes to the service Text Static. At the end of the modeling activity, the user selects the Deploy command to install the AppInventor on his device.

VI. GUI AUTOMATIC GENERATION

AppInventor involves the user in the composition of the application logic. As this application needs the composition of the multiple services provided by the mobile phone it includes the modeling of the GUI which involves the operations like pull-down menus, buttons, scrolling, iconic images etc. The composition of the user interface is not an easy task for user, so we come up with the solution as following: for the services provided by mobile phone, the GUI is generated using an XML description provided by the AppInventor; for Web services, it is automatically generated starting from the WSDL.

VII. ASSESSMENT

In AppInventor we have provided the platform to the end user to create his own application. To evaluate the usability of AppInventor we have studied some implementation(Related Work) regarding this application which shows the improvement of GUI in an AppInventor application. The evaluation proposed in this paper concerns the use of mobile computing features of both existing implementation corresponding to AppInventor(MicroApp) and App Inventor related to the development of applications executing Web services integrated with native device features (like mobile phones,tablet etc.), and exploiting the user context.

VIII. RESULTS

In this section, we have presented some snapshots of the application.

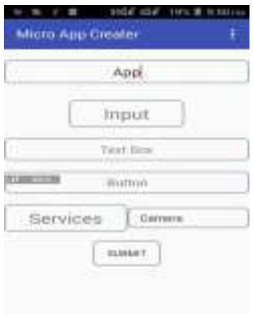


Fig. A

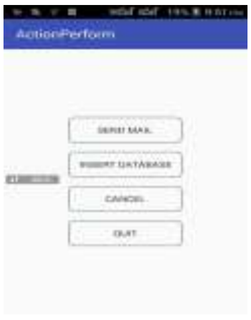


Fig. B

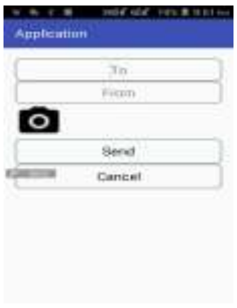


Fig. C

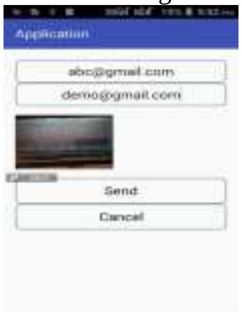


Fig. D

Figure A shows the GUI provided to the end user using which he can create his own application. We have provided very simple approach that can be useful to any user having no technological skills. Figure B shows the service provided by the application like camera, database; where we can select the required service. In Figure C and Figure D execution of service camera is shown; where we have used email as a service. This paper studies the detailed implementation of AppInventor application. Also we perform the analysis of the parameter time, performance and user interface. The chart is shown below:

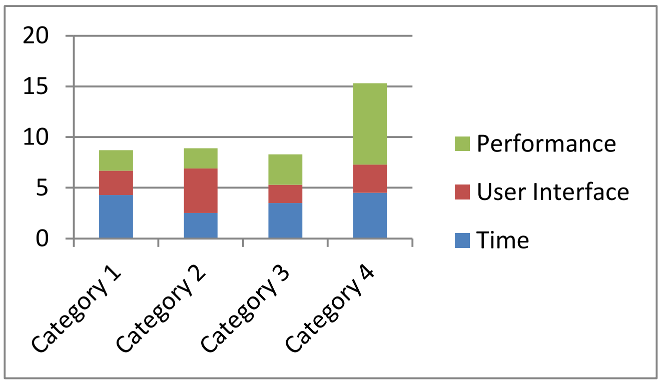


Fig 4. Performance Analysis

As much as the time required to execute the application is maximized it will affect the performance of the system. If the system provides the improved and user friendly GUI, it results into the efficient system which requires no technological skills to create application.

IX. FINAL REMARKS

In this paper we presented a mobile application and development process that support the user in the visual composition of customized applications for mobile devices. AppInventor which supports the generation of pervasive mobile applications, directly on the smart phone. AppInventor are graphically developed by composing native device features with Web services, domotics and sensor management services.
Future work will be devoted to enable the users to tweak the GUI to match their needs and to port AppInventor on other Mobile Operating Systems.

REFERENCES

[1] J. Danado and F. Patern` o, "A Prototype for EUD in Touchbased Mobile Devices," in IEEE Symp. on Vis. Languages and Human-Centric Computing (VL/HCC), 2012, pp. 83–86.
[2] "Puzzle: A Visual-Based Environment for End User Development in Touch-based Mobile Phones," in HCSE, 2012, pp. 199–216.
[3] N. Tillmann, M. Moskal, J. de Halleux, and M. Fahndrich,"TouchDevelop: Programming Cloud-connected Mobile Devices via Touchscreen," in SIGPLAN Symp. on New ideas, new paradigms, and reflections on prog. and softw. (ONWARD). ACM, 2011, pp. 49–60.
[4] J. Danado, M. Davies, P. Ricca, and A. Fensel, "An Authoring Tool for User Generated Mobile Services," in Conf. on Future Internet, 2010, pp. 118–127.
[5] M. Davies, F. Carrez, D. Urdiales, A. Fensel, M. Narganes,and J. Danado, "Defining User-generated Services in a Semantically-enabled Mobile Platform," in Intl. Conf. on Inf. Integration and Web-based App. & Services (iiWAS). ACM, 2010, pp. 333–340.
[6] D. Skrobo, HUSKY: A Spreadsheet for End-User Service Composition. http://www.fer.unizg.hr/ download/repository/DanielSkrobo KvalifDrIspit HUSKY.pdf, 2011.
[7] D. D. Hoang, H.-y. Paik, and B. Benatallah, "An Analysis of Spreadsheet-based Services Mashup," in Australasian Conf. on Database Tech. (ADC), 2010, pp. 141–150.
[8] J. Wong and J. I. Hong, "Making Mashups with Marmite:Towards End-user Programming for theWeb," in SIGCHI Conf. on Human factors in computing systems (CHI). ACM, 2007, pp.1435–1444.
[9] R Ahmed, "Visual Languages: A New way of Programming", in Malaysian Journal of Computer Science, 1999.
[10] Rita Francese, Michele Risi, Genoveffa Tortora, senior Member, IEEE, and Maurizio Tucci, "Visual Mobile Computing for Mobile End-Users," in IEEE Trans. On mobile computing, vol. -, no. -, APR 2015