Developing Context-Based Applications Using Visual Programming

Case Studies on Mobile Apps and Humanoid Robot Applications

Martin Zimmermann
Department of Economics
Offenburg University
Offenburg, Germany
e-mail: m.zimmermann@hs-offenburg.de

Abstract— **Sensors and actuators enable creation of context-aware applications in which applications can discover and take advantage of contextual information, such as user location, nearby people and objects. In this work, we use a general context definition, which can be applied to various devices, e.g., robots and mobile devices. Developing context-based software applications is considered as one of the most challenging application domains due to the sensors and actuators as part of a device. We introduce a new development approach for context-based applications by using use-case descriptions and Visual Programming Languages (VPL). The introduction of web-based VPLs, such as Scratch and Snap, has reinvigorated the usefulness of VPLs. We provide an in-depth discussion of our new VPL based method, a step by step development process to enable development of context-based applications. Two case studies illustrate how to apply our approach to different problem domains: Context-based mobile apps and context-based humanoid robot applications.**

Keywords-Context-based Services; Sensors; Actuators; Mobile Applications; Location-based Services; Robot Applications; Humanoid Robots; Visual Programming.

I. INTRODUCTION

The main privilege of context-aware applications is to provide tailored services by analyzing the environmental context, such as location, time, weather condition, and seasons, and adapting their functionality according to the changing situations in context data without explicit user interaction. For example, mobile devices can obtain the context information in various ways in order to provide more adaptable, flexible and user-friendly services. In case of a tourist app, a tourist would like to see relevant tourist attractions on a map together with distance information, depending on its current location. Human robots, require sensors to gather information about the conditions of the environment to allow the robot to make necessary decisions about its position or certain actions that the situation requires. As a consequence, context-aware applications can sense clues about the situational environment making applications more intelligent, adaptive, and personalized.

Sensors and actuators as part of devises enable creation of context-aware applications in which applications can discover and take advantage of contextual information, such as user location, nearby people and objects. A general definition of context was given by Dey and Abowd [1]: “Any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.”

Categories of context information that are practically significant are [1]:

- Environmental Context: Include all the surrounding environmental conditions of current location (like air quality, temperature, humidity, noise level and light condition).
- Temporal Context: Consists of temporal factor such as current time, date, and season of the year.
- Personal (identity) Context: Specifies user’s characteristics and preferences like name, age, sex, contact number, user’s hobbies and interest.
- Spatial context: Involves any information regarding to position of entity (person and object) for instance orientation, location, acceleration, speed.

Combination of spatial, temporal, activity and personal contexts makes the primary context to understand the current situation of entities, these types of contexts can response basic question about when, where, what, who.

Visual Programming Languages (VPL) and hybrid visual programming languages are considered to be innovative approaches to address the inherent complexity of developing programs [2][3]. In this work, we introduce an in-depth discussion of a new VPL based method, to enable even programming beginners the creation of context aware applications.

The rest of the paper is organized as follows: Section 2 introduces visual programming concepts, especially flow-based and object-oriented approaches. To illustrate how to apply our approach to different problem domains, context-based mobile apps and context based humanoid robot applications serve as case studies in Sections 3 and 4.

Copyright (c) IARIA, 2024. ISBN: 978-1-68558-100-8

Finally, the limitations of the VPL approach, as well as directions for future research are presented in Section 5.

II. Visual Programming

VPLs let users develop software programs by combining visual program elements, like sensor and actuator objects, loops or conditional statements rather than by specifying them textually [1].

A. *VPL Concepts*

A comprehensive analysis of various VPLs including the strengths and weaknesses of VPLs, as well as guidelines to choose the most suitable VPL for the task in hand is described in [2]. There are two popular categories of VPLs: Flow-based VPLs and object-oriented VPLs.

In case of object-oriented VPLs, visual program elements are based on an object-oriented paradigm, i.e., decomposition of a system into a number of entities called objects and then ties properties and function to these objects. An object's property can be accessed only by the functions associated with that object but functions of one object can access the function of other objects in the same cases using access specifiers. Figure 1 shows the visual elements for a simple function call, following the representation of visual elements used in MIT AppInventor [4]. Objects, method calls, arguments and results of method calls are represented by visual elements with different shapes and colors. Clicking a button, touching a map, and tilting the phone are examples for user-initiated events.

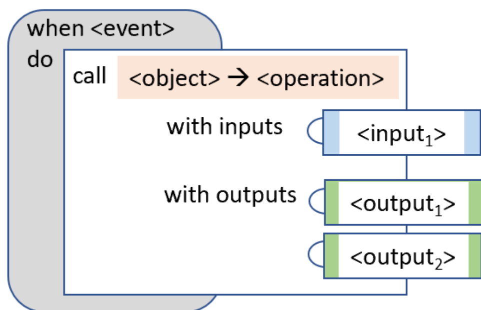

Figure 1. Object-oriented Visual Programming.

In general, flow-based VPLs offer different categories of elements, e.g., function calls, as well as control elements, like conditional statements. Programs are developed by placing them one after the other. For a parallel execution, more than one element may be used. Figure 2 illustrates the basic idea of a flow-based VPL-based program following the representation of visual elements used in Choregraphe [5]. First, the function f1 is called, then f2 and f3 are executed in parallel. Function elements are connected by using entry and exit ports. This is similar to BPMN, which stands for Business Process Model and Notation. BPMN is a standardized graphical modeling language used to represent and visualize business processes [6]. It is a widely adopted industry standard for modeling and documenting business processes, as it provides a consistent and easy-to-understand visual representation of a process. Flow elements are elements that connect with each other to form business workflows similarly to the visual building blocks of Choregraphe.

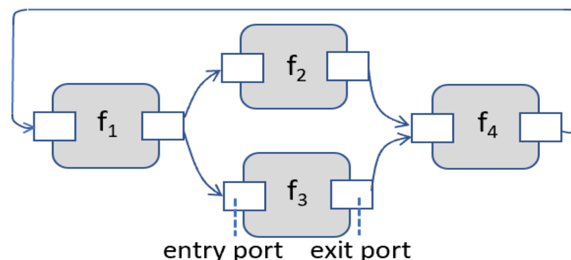

Figure 2. Flow-based Visual Programming.

Examples for flow-based systems are Flowgorithm [7], Microsoft VPL [8], and Choregraphe [5]. Flowgorithm is a general-purpose VPL that can be used to create flowchart representations of computing algorithms. It can translate a visual flowchart into eleven different textual programming languages, including C++ and Java. Flowgorithm is language independent but is not platform-independent; it is available only for Microsoft Windows operating system. Although it is not open source but is free for use.

Microsoft VPL is a programming environment based on graphical data-flows. It is aimed at engaging hobbyist programmers, as well as professionals. Furthermore, novice programmers can use it to learn programming, and experienced programmers can employ it for rapid prototyping. Microsoft VPL is a part of Microsoft Robotics Developers Studio (MRDS) used to develop software to guide robots. Choregraphe [5] represents also a flow-based visual programming environment which offers visual program elements for to easily develop complex robot applications, but Choregraphe also offers limited operations for the localization, mapping, as well as simple navigation functions.

B. *Use Case Driven Development*

Requirements engineering is done in two steps: Development of a use case diagram and specification of the use cases (each with input, output, steps). Based on the developed use cases, the VPL based application coding process is also use case driven and done in three steps:

- Development of a user interface for each use case (optional part).
- Selection of components (e.g., location sensor, QR code scanner, etc.).
- Flow-based or object-oriented visual programming

In Section 3, we show how our approach is applied to two very different problem domains, namely context-based mobile applications and robot applications.

Copyright (c) IARIA, 2024. ISBN: 978-1-68558-100-8

III. Case Study: Humanoid Robot Applications

In this section, we present a case study of a context-based application for a humanoid robot. First, the sensors and actuators are analyzed, then the visual program elements are explained and finally the VPL based implementation of a context-based use case is described.

A. *Sensors and Actuators*

As an example, we use the popular humanoid robot Pepper (by SoftBank Robotics) [9][10], see Figure 3, a wheeled humanoid robot with sensors and actuators, i.e., torso, a head, two arms, with 20 degrees of freedom for motion in the body (17 joints for body language) and three omnidirectional navigation wheels to move around smoothly. It contains a set of various sensors to allow it to perceive objects and humans in its environment [11]:

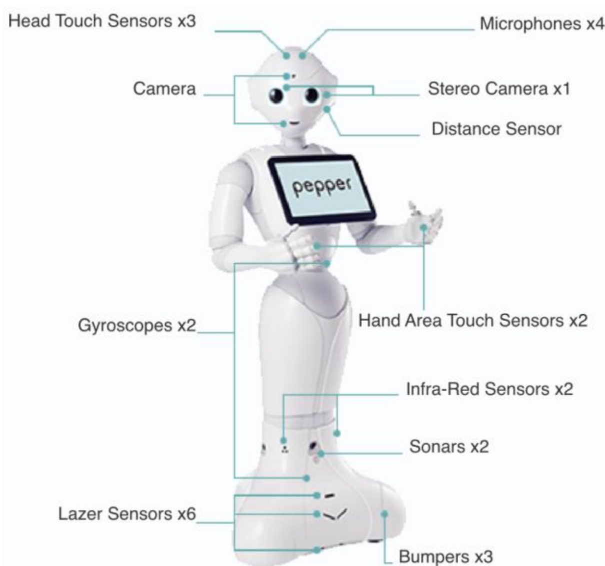

Figure 3. Sensors of the Pepper platform.

Pepper is able to localize a person talking to it, can distinguish multiple faces, determine eye contact or even recognize and react to basic emotions of the person it is talking with. Additionally, the humanoid robot is able to recognize someone's emotion not only by voice, but also by parameterizing facial expressions of interlocutors by using machine vision.

B. *Visual Program Elements*

Choregraphe is a visual programming environment for the Pepper platform. It offers visual program elements for different categories to easily develop complex robot applications, but Choregraphe also offers limited operations for the localization, mapping, as well as simple navigation functions [11].

Table I illustrates examples for the different categories of visual program elements, e.g., visual building blocks for speech creation, camera actions, or human face detection. Additional AI-based building blocks returns the gender, the age or the detected facial expression of the person in front of the robot. Building blocks include also logic functions and conditional statements, i.e., a condition and stimulate the then or else outputs depending on the boolean value of the condition.

TABLE I. Visual Program Element examples.

Visual element	Category		
	Sensor	Actuator	Other
AnimatedSay (Animated Say)		Speech	
Face Detect (Face Detection)	Human Detection		
GetGender (Get Gender)	Human Understanding		
If statement (IF)			Logic

C. *Context-based Application: "Recognize and Greet People Scenario"*

In the following, we introduce a simple use case "Recognize & Greet People" as an example of a context-based application. The robot greets a person standing in front of it (after the robot has identified a face). Optionally (if questions are asked), the robot answers a question in the second use case "answer question" (not shown). The use case specification for the base use case is described in Table II. Based on a loop, four steps have to be executed: Detect face, determine gender, determine age and finally an animated say depending on the results of the previous blocks.

TABLE II. *Use Case: "Recognize & greet people".*

	Recognize and greet people
Input	Person in front of a robot
Steps	Loop 1: Detect face 2: Determine gender 3: Determine age 4: if (gender = female or gender = male) and (age < 30) Say "Hi, how can I help you" else if (gender = female) and (age >= 30) Say "Good Morning Madame, how can I help you" else if (gender = male) and (age >= 30) Say "Good Morning Sir, how can I help you"
Output	Voice output (depending on gender and age of face)

Copyright (c) IARIA, 2024. ISBN: 978-1-68558-100-8

Based on the results of the requirements engineering, the implementation is also use case driven in three steps:

- Selection of visual program elements: Sensors, actuators and control elements
- Setting parameters of the visual programming building blocks (e.g., set the language of the “Animated Say” building block or the text to be spoken).
- Connection of visual programming building blocks, i.e., connecting their inputs and outputs, e.g., the output of “Face Detection” element must be connected to the input of the “Get Gender” element

Figure 4. Sensors and Actuators for the Use Case.

For the use case “Recognize & Greet People”, the following sensors and actuators are required: Face Detection, GetAge, GetGender, and AnimatedSay (Figure 4). For example, GetGender returns the gender of the person in front of the robot. It is possible to set up the confidence threshold and the timeout.

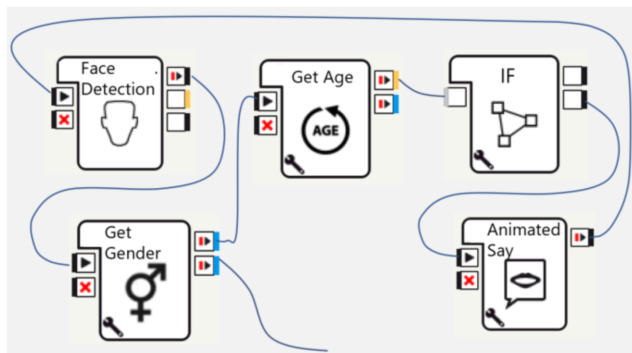

Figure 5. Flow-based implementation of the Use Case.

Figure 5 illustrates the flow-based implementation of the use case.

IV. CASE STUDY: MOBILE APPLICATIONS

In this section, we present a case study of a location-based mobile application (spatial context). First, the sensors and actuators are analyzed, then the visual program elements are explained and finally the VPL based implementation of a concrete location-based service is described.

A. *Sensors and Actuators*

Sensors in mobile devices measure various environmental parameters, such as ambient air temperature and pressure, illumination, and humidity (Figure 6). This includes orientation sensors, magnetometers, but also barometers, photometers, and thermometers. Actuators mainly perform vibration-related functions, such as vibration and sound generation.

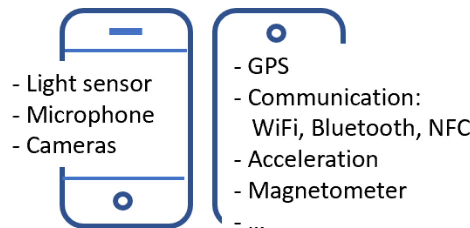

Figure 6. Sensors and Actuators of Mobile Devices.

B. *Visual Program Elements*

We use App Inventor [4] and Thunkable [12], which are both cloud-based visual development environments for mobile applications (Android and iOS). App Inventor and Thunkable provide the application developer with many different components to use while building a mobile app. Properties of these components such as color, font, speed, etc. can be changed by the developer. Available element categories are user interface elements, media, storage, location-based services etc. Elements can be clicked on and dragged onto the development screen area.

There are two main types of components: Visible and non-visible. Visible components such as buttons, text boxes, labels, etc. are part of the user interface whereas non-visible components such as the location sensor, QR Code scanner, sound, orientation sensor are not seen and thus not a part of the user interface screen, but they provide access to built-in functions of the mobile device.

TABLE III. VISUAL PROGRAM ELEMENT EXAMPLES.

Visual element	*Sensor*	*Actuator*	*Other*
Location Sensor	Get location information		
BLE	Connect to BLE devices		
Vibration		Vibrate for a specified time	
Loop			Iterate through elements

Table III illustrates examples for the different categories of visual building blocks. The location sensor provides location information, including longitude, latitude, altitude (if supported by the device), speed (if supported by the device). The Bluetooth Low Energy (BLE) component allows an application to find and connect to BLE devices and to communicate directly with them. The vibration actuator will vibrate the device for a specified time unit. The foreach element applies a set of functions to each element of a list, e.g., a list of tourist locations, part of a tourist app.

C. *Context-based Application: Location-based Service Scenario*

The use case in Table IV describes a location-based service [13] pattern in terms of input, output and steps to be executed. The template can be applied for instance to visualize some tourist attractions and the current position of a mobile user on a map. Filters are used to display certain tourist attractions, e.g., museums.

TABLE IV. USE CASE TEMPLATE "SHOW OBJECT(S)".

Use Case	*Show <object(s)> on a map*
Input	Filter
Steps	1: Determine the current geo position of the user 2: Show a map with the user's current position as the center point 3: Search the <object(s)> according to the specified filter (in a list of <object>) if found → Create marker(s) for the <object(s)>
Output	Map with markers: → marker for the current position of the user → marker(s) representing the <object(s)>

The implementation based on an object-oriented VPL, like MIT AppInventor, follows three steps:

- Selection of visual program building blocks: User-interface elements, sensors, actuators and control elements (e.g., if, switch elements).
- Definition of events (e.g., when button click, on map loaded)
- Calling methods (e.g., calling a method to get the current location of a user).

Event handler blocks specify how a program should respond to certain events. After, before, or when the event happens can all call different event handlers. There are two types of events: user-initiated and automatic.

Clicking a button, touching a map, and tilting the phone are user-initiated events. Sprites colliding with each other or with canvas edges are automatic events. Timer events are another type of automatic event. Sensor events function also as user-initiated events. For example, orientation sensor, accelerometer, and location sensor all have events that get called when the user moves the phone in a certain way or to a certain place.

Figure 7 shows the visual elements of the event-based programming part for the use case in Table IV, a simple location-based service. Objects, method calls, arguments and results of method calls are represented by visual elements with different shapes and colors. In a first step the current position of a user is determined by calling the method GetCurrentLocation. The resulting values (latitude and longitude) are used in the next step for the specification of the map center (two set operations). By calling the method addMarker, a marker is created in a third step. The arguments for the last method call are again visual elements (previously calculated values for the current latitude and longitude of the user).

Finally, a corresponding marker is generated for all objects of a list (by using a "for each item loop"), according to the specification in the use case description (Table IV).

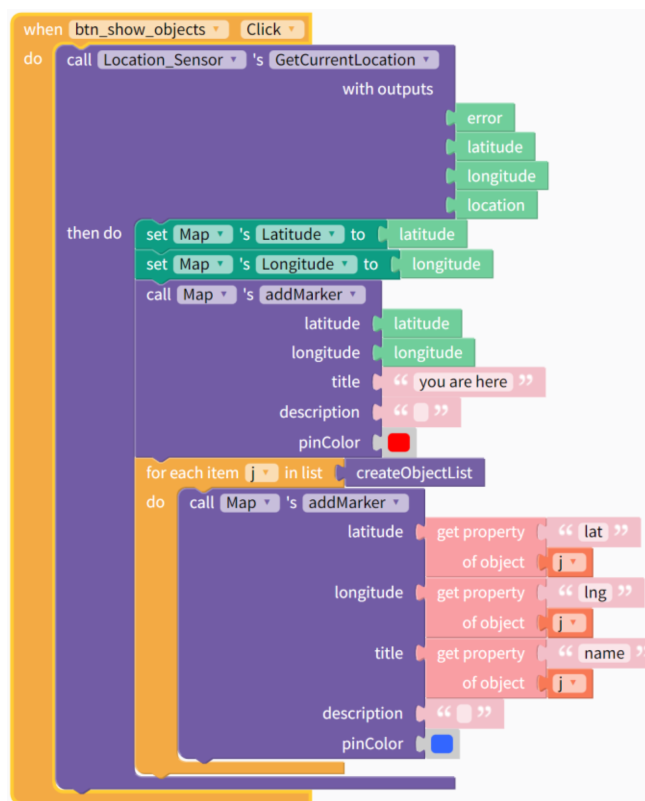

Figure 7. Implementation of a Use Case by Using an Object-Oriented VPL.

Creation of the object list could be based on a local list (as part of a mobile app) or a cloud-based object list. The creation / access to an object list in encapsulated in a separate function createObjectList. Finally, the objects behind a marker have to be visualized. For example, exhibits, like paintings in a gallery could be equipped with qr codes.

Copyright (c) IARIA, 2024. ISBN: 978-1-68558-100-8

Object visualization in this case could mean to create a link to a video (painter explaining interesting background information) or a link to allow a tourist buying a print.

V. CONCLUSION

The main privilege of context-aware applications is to provide an effective, usable, rapid service by considering the environmental context (such as location, time, weather condition, and other attributes) and adapting their functionality according to the changing situations in context data. Use cases and visual programming are particularly well suited for programming beginners. However, visual programming environments are increasingly used in demanding problem domains, e.g., Internet Of Things (IoT) applications [14]. The development of use cases (in the sense of requirements engineering) as the starting point of an context-based application project has proven to be very advantageous.

Benefits of VPLs are short development times, low costs and increased efficiency and productivity [15]. VPL tools enable users with low technical skills to develop advanced software. Both VPL approaches, flow-based and object-oriented programming have their application areas. If parallel activities in particular are to be programmed, then flow-based systems are usually better suited. Our experience is that object-oriented VPLs seem to be more intuitive (especially for with low technical skills) due to the visual representation of objects, methods and the set and get operations.

A main drawback of the used programming environments is the identification and handing of runtime errors due to the lack of integrated debugging functions. However, our use case centered approach leads normally to manageable runtime error because each use case is developed and tested as a separate unit.

Future work will focus on the development of patterns and model-driven development [16]. Patterns are a well-known concept in the traditional software engineering. An architectural pattern is a general, reusable solution to a commonly occurring problem in software architecture. An architectural pattern becomes a reusable solution for a common set of problems in software development, addressing issues like high availability, performance, and risk minimization. Additionally, we focus on development of a repository of use case templates and visual code templates to improve design and implementation of context-based applications [17].

REFERENCES

[1] A. K. Dey and G. D. Abowd, "Towards a Better Understanding of Context and Contextawareness," CHI 2000 Workshop on The What, Who, Where, When, Why and How of Context-awareness, pp. 1–6, 2000.

[2] M. Idrees and F. Aslam, "A Comprehensive Survey and Analysis of Diverse Visual Programming Languages," VFAST Transactions on Software Engineering, vol.10, no. 2, pp. 47–60, 2022.

[3] R. Daskalov, G. Pashev, and S. Gaftandzhieva, "Hybrid Visual Programming Language Environment for Programming Training," TEM Journal, vol. 10 Issue 2, pp. 981–986, 2021.

[4] MIT App Inventor. https://appinventor.mit.edu, [retrieved: September, 2023].

[5] Choregraphe, http://doc.aldebaran.com/2-4/software/choregraphe/, [retrieved: October, 2023].

[6] BPMN, https://www.omg.org/bpmn, [retrieved: October, 2023].

[7] Flowgorithm, http://www.flowgorithm.org/, [retrieved: October, 2023].

[8] Microsoft vpl, https://msdn.microsoft.com/enus/library/bb483088.aspx, [retrieved: October, 2023].

[9] Pepper, http://doc.aldebaran.com/2-4/home_pepper.html [retrieved: October, 2023].

[10] C. Gómez, M. Mattamala, T. Resink, and J. Ruiz-Del-Solar, “Visual SLAM-Based Localization and Navigation for Service Robots”: The Pepper Case. In Robot World Cup; Springer: Cham, Switzerland, pp. 32–44, 2018

[11] A. M. Marei, et al., “A SLAM-Based Localization and Navigation System for Social Robots: The Pepper Robot Case”, Machines 2023, 11(2), pp. 47–60.

[12] Thunkable. https://thunkable.com, accessed: 2023-07-10.

[13] T D’Roza and G Bilchev, "An overview of location-based services," BT Technology Journal, vol. 21, no. 1, pp. 20–27, 2003

[14] M. Silva, J. P. Dias, A. Restivo, and H. S. Ferreira, “A Review on Visual Programming for Distributed Computation in IoT”, Springer Nature Switzerland AG 2021, M. Paszynski et al. (Eds.): ICCS 2021, LNCS 12745, pp. 443–457, 2021

[15] D. Pinho, A. Aguiar, and V. Amaral, "What about the usability in low-code platforms? A systematic literature review", Journal of Computer Languages, Volume 74, pp. 1959–1981, 2023

[16] Md. Shamsujjoha, J. Grundy, Li Li, H. Khalajzadeh, and Q. Lu, Developing Mobile Applications Via Model Driven Development: A Systematic Literature Review, Information and Software Technology, Volume 140, December 2021.

[17] M. Zimmermann, “Location and Object-Based Mobile Applications”, UBICOMM - International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies, pp. 34-39, 2023.