Picture-Driven User Interface Development for Applications on Multi-platforms

Vinh-Tiep Nguyen[1], Minh-Triet Tran[1], and Anh-Duc Duong[2]

[1] Faculty of Information Technology, University of Science, VNU-HCM, Vietnam
[2] Faculty of Software Engineering, University of Information Technology, VNU-HCM, Vietnam
{nvtiep,tmtriet}@fit.hcmus.edu.vn, ducda@uit.edu.vn

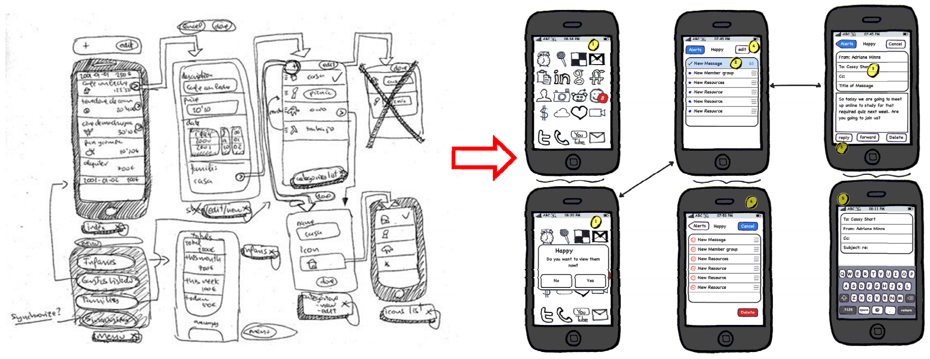
Abstract. Graphical user interfaces are usually first sketched out manually as hand drawing pictures and then must be realized by software developers to become prototypes or usable user interfaces. This motivates our proposal of a smart CASE tool that can understand hand drawing sketches of graphical user interfaces, including forms and their navigations, then automatically transform such draft designs into real user interfaces of a prototype or an application. By using the ideas of modeling and model-transformation in model driven engineering, the authors also propose a mechanism to generate graphical user interfaces as forms targeting different platforms. Experimental results show that our sketch recognition to understand hand drawing graphical user interfaces can achieve the accuracy of 97.86% and 95% in recognizing 7 common UI controls and arrows for navigation respectively. Our model transformation engine can generate user interfaces as forms for applications on 3 different platforms of mobile devices, including Windows Phone, Android, and iOS. This approach follows the trend to develop a new generation of smart CASE tools that can understand and interpret conceptual software design models into concrete software elements and components to assist the software development process in a natural way.

Keywords: picture-driven, graphical user interface, code generation, mobile device, multi-platform.

1 Introduction

Graphical User Interface changed the way people interact with computers and computing systems more intuitively and attractively. By using GUI, a user can easily understand and follow the workflow of a business process in an application. In software development process, early phases are very important, especially in the phase of gathering software requirements. To understand and collect correct and enough information of requirements, developers usually sketch out key ideas, including main user interfaces and their navigations (c.f. Figure 1a), to discuss with customers.

Although there are different tools to assist developers design user interfaces, such utilities still do not have the capability to understand hand drawing user interfaces and the semantic for the navigations between forms and controls. To bridge the gap between user interface designs as sketches and concrete user interfaces, there is a practical need to transform instantly sketch ideas of user interfaces into concrete GUIs for software to avoid misunderstanding and to help developers capture exactly workflows of the system. This opens a new generation of computing, picture-driven computing[1]. Figure 1 demonstrates the idea of turning sketch user interfaces to software prototypes.



(a) Sketch user interface (b) Software prototype

Fig. 1. Idea of sketch user interface to software prototype

Moreover, software is often developed with various distributions for multi-platforms with the same user interfaces and functions. It would be time consuming to redesign the user interface of the same software when a developer wants to port that application to another platform. Thus it would be more convenient for developers to automatically transform an abstract UI design that is independent from any specific platform to a concrete realization of the design for a particular platform, such as Windows Phone, Android, or iOS.

In this paper, we propose our idea that to develop a model-driven method to realize user interfaces targeting multi platforms from hand drawing sketches of graphical user interfaces. By understanding the semantic of an image, in this case, a sketch of a user interface design, our system can automatically generate a concrete user interface with required source code for an application on a specific platform. Our idea can be used not only in software prototyping but also in assisting developers to migrate an application to another platform just by model transformations. The two main components to solve two problems in our proposed system are as follows:

- Sketch recognition: this component is responsible for recognizing symbols of UI controls used in the quick sketch design of software user interface. The component also determines the spatial structure of these symbols to capture the topology order and relationship between UI controls, i.e. which object is a control, which object is a container, and what event will be generated when a user interacts with a certain UI object.

- Model-driven code generator: from the spatial structure of symbols, an Abstract User Interface (AUI), a platform-independent UI model of an application, is generated. This AUI model plays the role of the common model to generate different Concrete User Interfaces (CUIs), platform-dependent user interfaces of the same application.

The main contributions of our proposal are to propose a histogram-of-gradient based feature for sketch recognition; to decompose the whole sketch image into separable forms and controls with navigation; and to develop a framework for multi-platform code generation based on model-driven architecture.

The content of the paper is as follows. In Section 2, the authors briefly review some approaches of sketch recognition problem and UI control code generation. Our proposed system and experimental results are presented in Section 3 and 4 respectively. Conclusions and future work are discussed in Section 5.

2 Background and Related Work

2.1 Sketch Recognition

Sketch recognition is a special case of visual object recognition. There are some common approaches for sketch recognition, such as template matching or local feature based matching. Template matching approaches [2] [3] use color information of a template as the main factor to determine the similarity between the template and an extracted pattern from a source image. Edge-based template matching[4] can work well with a low-texture object such as a sketch but it is not robust with some small changes of the sketch.

Local feature based approaches such as SIFT[5], SURF[6] are robust with scale and rotation transformations but these methods are only effective when working with high-texture objects. For the special case of objection recognition problem, a sketch object often has extra useful information such as its shape and directions of movement in the drawing of the sketch. This information can be extracted based on edge features. HoG[7] is a typical feature descriptor that counts the occurrences of gradient orientations in localized portions of an image. This feature is well known in pedestrian detection problem in static image. To work with various types and shapes of sketches, machine learning is a very common approach to think about. There are many learning algorithms and learning models such as: SVM[8], Neural Network[9], Bag of Features[10]. These algorithms often require a large dataset with high computational cost. So in this project, we try to make the implementation process as simple as possible.

2.2 Model Driven Code Generation for Multi Platform Applications

With the rapid development of different mobile platforms, an application can be developed with multiple distributions targeting multiple platforms. It would be waste of time and effort to redesign user interfaces of an application when it is migrated to a new mobile platform. Model-driven approach is one of the promising solutions to help developers in porting applications for multi platforms[11].

In Model-Driven Development (MDD[12]), a software is first designed with high level models that do not depend on any specific technical background. These models are then transformed into low level models that are dependent on a specific platform and linked to a particular technology[13]. By this way, it would be convenient to develop the same application on different platforms simply by transforming the abstract, high level models of the software into different concrete, low level models, even to the source code in a specific development environment.

To solve the problem of generating the same UI design (from sketch drawing) to various mobile platforms, we inherit the concept of Platform Independent Model and Platform Specific Model[13]. A Platform Independent Model (PIM) user interface is independent from any specific mobile platforms while a Platform Specific Model (PSM) is bound to a specific technology, such as Android, Windows Phone, or iOS.

3 Proposed Method

3.1 System Overview

The proposed process consists of three main phases: training phase using sketch image dataset, recognizing phase to identify drawn sketch, and code generating phase. The overview of our system is illustrated in Figure 2.

Training Phase: The input of this phase is a sketch dataset with multiple sketch images of common UI controls, e.g. forms, buttons, textboxes, combo boxes… Each sketch image is labeled corresponding to its UI control type. This module transforms a sketch image into a feature map. These maps are archived in the feature set to be used in next phase. Two feature maps are compared using a distance measure. In this work, Euclidean distance is chosen for simple implementation. Besides sketches of common UI controls, an arrow is proposed as a special sketch with many different characteristics to link a button to a form. Recognizing arrow must be invariant with shape and rotation transformation. We will discuss two algorithms to recognize these types of sketches in section 3.2.

Recognizing Phase: In this phase, new sketches drawn by user when designing graphic user interface are classified into corresponding UI control types. Each feature extracted from a new sketch is compared to trained feature set. Labels of UI new sketches are determined based on best matched sample in the training dataset.

Code Generating Phase: After recognizing a sketched UI, it is necessary to localize its position to determine its corresponding form. Based on its position, our system constructs the spatial structure of all controls that can be used to understand which object is a control, which object is a container, and what event will be generated. Finally, the code generator engine generates software prototype corresponding to the input sketched design targeting a specific platform, e.g. Windows Phones, Android, or iOS.

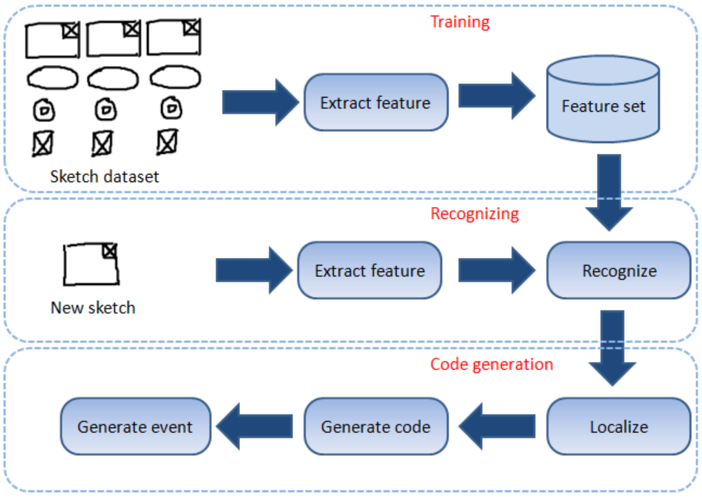


Fig. 2. Overview of the proposed system

3.2 Sketch Recognition Method

Training Phase

In this paper, we focus on recognizing some common UI control in graphic user interface design. There are seven types of controls: main form, tab control, text box, button, combo box, check box and radio button (Figure 3). Among these types, text box and button are very easy to be mistaken if user does not pay attention to some important differences. Button is drawn by smooth curves whereas text box is drawn by straight lines and sharp corners. To connect forms or tab controls together, we use arrow as a special type sketch. An arrow can have many directions, lengths and shapes as illustrated in Figure 4. It is very difficult to use the same method to recognize both graphic controls and arrow sketch. Therefore we propose two distinct algorithms to recognize these types of sketches to take advantages of specific constraints of each problem.

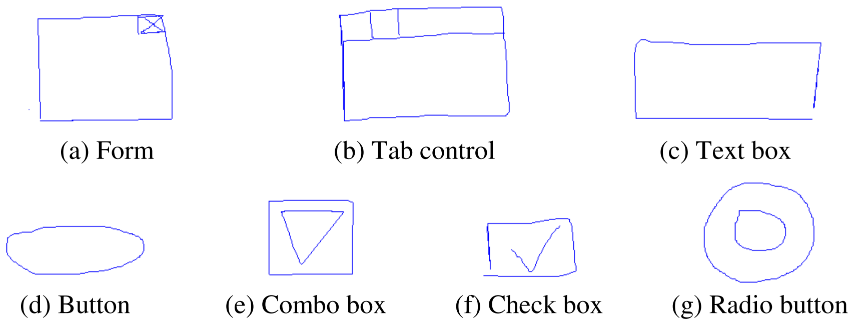


Fig. 3. Seven types of UI controls

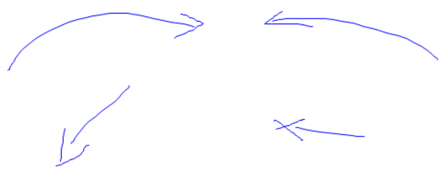


Fig. 4. Example of various directions, lengths, and shapes or navigation arrows

The main advantage of UI control sketch recognition problem is that it does not need to be invariant with rotation transformation. Inspired by the idea of HoG (Histogram of Gradients)[7], we propose our new simple feature map of sketch. Each sketch is represented as four 16x16 feature maps corresponding to four main directions $\{0, \pi/4, \pi/2, 3\pi/4\}$. This feature is used to compare with those of trained sketch samples in the dataset using Euclidean distance for simplicity and ease of implementation. The UI control recognition includes 3 steps:

- **Step 1-Compute Gradients:** To take advantage of order information of points in sketch, angle of gradient vector is computed for each point. Formula of tangent vector at the i^{th} point:

$$\alpha_i = \tan^{-1}\left(\frac{y_{i+w} - y_{i-w}}{x_{i+w} - x_{i-w}}\right)$$

where w is the width of tangle window. After this step, we have a set of (x_i, y_i, α_i) corresponding to each point of sketch and its angle. Spatial information and direction are combined to create feature maps in next steps.

- **Step 2- Normalize Coordinate:** Since UI controls have many different sizes, we normalized original coordinate into feature space to make the algorithm robust with various sizes of control. We use the following formula to transform coordinate:

$$x_{i\_norm} = \left(\frac{x_i - \mu_x}{\sigma_x}\right)\frac{h}{5} + \frac{h}{2},\ y_{i\_norm} = \left(\frac{y_i - \mu_y}{\sigma_y}\right)\frac{h}{5} + \frac{h}{2}$$

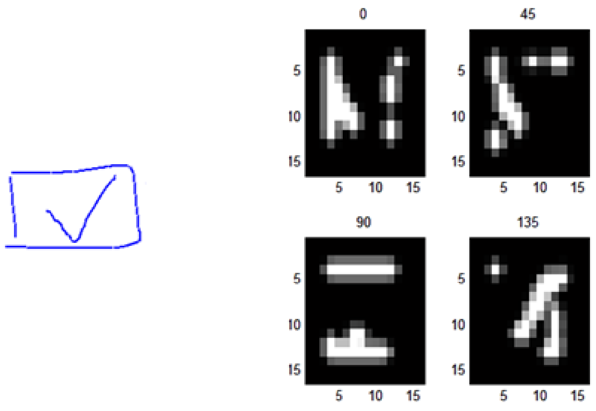
where

$$\mu_x = \frac{1}{n}\sum_{i=1}^{n} x_i, \mu_y = \frac{1}{n}\sum_{i=1}^{n} y_i,\ \sigma_x = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu_x)^2}, \sigma_y = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \mu_y)^2}$$

h=16 is size of the feature map.

- **Step 3- Compute Feature Maps:** Since this problem does not need to satisfy invariance with rotation and direction of sketch movement when drawing, only the value of orientation in range of $[0, \pi)$ is used. In this paper, we divide this range into 4 bin of orientation including $\{0, \pi/4, \pi/2, 3\pi/4\}$. Each orientation corresponds to a feature maps. Feature map bases voting principle of intension. After normalization, we accumulate intension of the angle into corresponding map.

For each point in line from (x_{i_norm}, y_{i_norm}) to (x_{i+1_norm}, y_{i+1_norm}), we accumulate intension of α_l corresponding to four maps of each orientation. Figure 5 illustrates an example of feature maps for a check box sketch.



(a) Check box sketch (b) Feature map for a check box sketch

Fig. 5. An example of feature extraction to feature map

As mentioned before, an arrow that links controls and forms is a special type of sketch in our system. It is very difficult to solve both two types of sketches at the same time. The system is then required to recognize sketches with unpredictable directions and shapes. However, an arrow also has some special characteristics so that we can use to recognize them easier. We inherit the idea of T. A. Hammond [14] to describe an arrow in a specification language to simplify sketch recognition user interface:

- An arrow is formed by three distinct segments. A distinct segment contains all point which has local angle approximate to π. In case a segment has a point that its local angle differs with straight angle, it is separated into two segments. Figure 6 shows an example of segment was separated into two parts.
- Three end points of segments intersect in the same position.
- The longest segment stays at the middle of other ones. All of arrows in Figure 4 satisfy these properties.

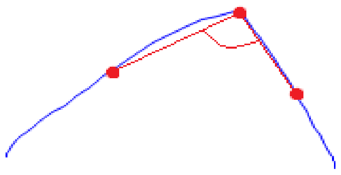


Fig. 6. A segment was separated into two segments

Recognizing Phase
To assist a developer in real-time manner to illustrate how the real user interface looks like as soon as the developer sketches out the draft design of a graphical user interface, we

When the developer designs a form or a set of forms with navigations (denoted by arrows), he or she usually draws on the drawing pad or a sheet of paper only one UI control or a navigation arrow at a time. There is usually a short pause, or idle state, between the drawing of two consecutive UI controls or arrows. Therefore, our system continuously monitors the on-going drawing to detect idle states between the sketches of two UI controls or arrows. By this way, our system can extract a new sketch corresponding to a single UI control or an arrow as soon as the developer finishes drawing it.

When a new sketch is extracted from the current UI design drawing, the system first verifies if it is an arrow, a navigation link between UI controls. If not, the system then continues to pass that new sketch into its UI control recognition module. The sketch is then transformed to its corresponding feature map to be comparedwith the feature maps of trained samples. For simplicity of implementation, we use Euclidean metric to compute distance between feature maps.

3.3 Code Generating Phase

To generate source code of user interfaces for an application targeting different platforms, we propose a framework following Model Driven Architecture (MDA) to easily transform source code from a platform to another one.

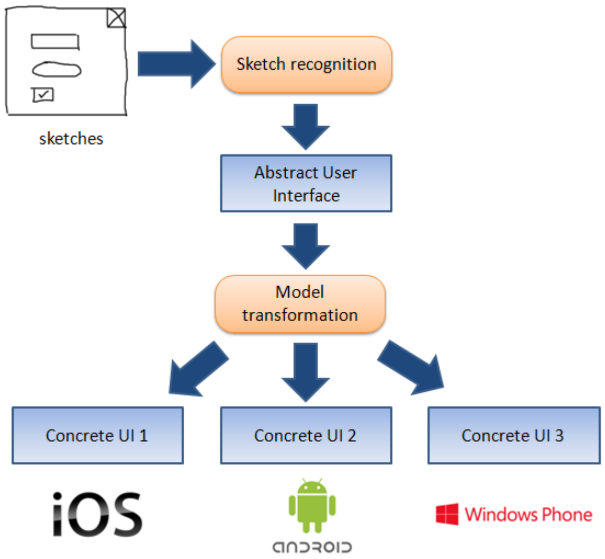


Fig. 7. The overview of code generating phase for multiple platforms

In a model-driven approach, a high level model is required because it is the common model that plays the role of a bridge between different low level models targeting various platforms and technologies. Figure 7 illustrates the overview of our proposed code generating phase for user interfaces on different platforms from a single hand drawing sketch.

We reuse the platform-independent model of user interfaces for application on mobile devices proposed by C. K. Diep et.al[15]. The output of the sketch recognition process is not a concrete user interface that is dependent on a specific mobile platform but an Abstract User Interface (AUI[15]). This model is then transformed into different Concrete User Interfaces (CUIs [15])

4 Experimental Results

For the sketch recognition phase, we conduct two experiments to testing the accuracy of arrow recognition and UI control recognition algorithms. For the proposed arrow recognition algorithm, we create a test set of 100 samples (50 positive and 50 negative tests). The confusion matrix is illustrated in Table 1. We have false positive rate is 4/50 = 8% and false positive is 1/50 = 2%. The total accuracy is (46+49)/100=95%.

Table 1. Confusion matrix for arrow sketch recognition

	Negative	**Positive**
Negative	46	4
Positive	1	49

For UI control recognition, we conduct the experiment on 420 samples divided equally into 7 types of controls. 280 samples are used for training and 140 samples are used for testing. The accuracy of the proposed method is 137/140 samples (97.86%).The confusion matrix for UI control recognition is shown in Table 2. We can see that, a text box or a radio button can be misclassified as a button (3/40 cases).

Table 2. Confusion matrix for UI control sketch recognition

		Recognized control						
		Form	Tab control	Text box	Button	Combo box	Check box	Radio button
Actual control	Form	20	0	0	0	0	0	0
	Tab control	0	20	0	0	0	0	0
	Text box	0	0	18	2	0	0	0
	Button	0	0	0	20	0	0	0
	Combo box	0	0	0	0	20	0	0
	Check box	0	0	0	0	0	20	0
	Radio button	0	0	0	1	0	0	19

Figure 8 illustrates a demonstration of converting from sketch to a GUI. A sketch is captured by a special device such as stylus pen, tablet (c.f. Figure 8a). The current system can only detect and align which region is label with hand written character.

However, at the meantime, the system does not recognize of a label. A label control is added to the interface but its content must by edited manually by developer. It should be noticed that developer is required to assign text content of the label in an abstract user interface only. Then, this model will be transformed into different CUIs without any modification of the developer.

The source code of graphic user interface is generated automatically for an application targeting iOS, Windows Phone, Android (c.f. Figure 8b,c,d). We also propose an algorithm to align controls of the same type in a specific group to have a better layout. Our system currently supports UI source code generation for 3 main mobile platforms: iOS, Windows Phone and Android.

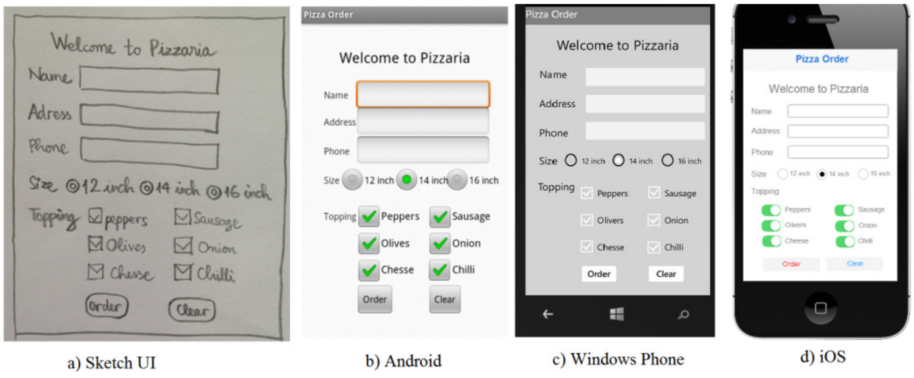

Welcome to Pizzaria
Name
Adress
Phone
Size 12 inch 14 inch 16 inch
Topping peppers Sausage Olives Onion Chese Chilli
Order Clear
Pizza Order
Welcome to Pizzaria
Name
Address
Phone
Size 12 inch 14 inch 16 inch
Topping Peppers Sausage Olives Onion Chesse Chilli
Order Clear
Pizza Order
Welcome to Pizzaria
Name
Address
Phone
Size 12 inch 14 inch 16 inch
Topping Peppers Sausage Olivers Onion Chesse Chilli
Order Clear
Pizza Order
Welcome to Pizzaria
Name
Address
Phone
Size 12 inch 14 inch 16 inch
Topping Peppers Sausage Olivers Onion Cheese Chilli
Order Clear
a) Sketch UI
b) Android
c) Windows Phone
d) iOS

Fig. 8. An example of converting a hand drawing UI sketch into three concrete GUIs for an application

5 Conclusion and Future Work

In this paper, we propose a smart tool that can understand a typical hand drawing sketch of a GUI draft design and realize it into concrete user interfaces for a real application on different mobile platforms, including Windows Phone, Android, and iOS. In our system, we propose an algorithm for sketch recognition, an algorithm for separating a sketch image into independent components, and a model-based framework to generate code of user interfaces for applications on multiple mobile platforms.

Currently our solution supports 7 common UI controls, including forms, tab controls, textboxes, buttons, combo boxes, check boxes, and radio buttons. Our sketch recognition module can also identify arrows liking between forms to generate appropriate UI navigations. More UI controls can be trained and added into the sketch recognition module to enrich the usability of our proposed method. With the application of model-driven approach, we utilize the Abstract User Interface and Concrete User Interface to enable our system targeting various mobile platforms. Furthermore, more sets of transformation rules can be added so that our system can support more mobile platforms. For more convenience, we also integrate hand written recognition module to recognize label automatically instead of manually editing by user.

References

1. Hardesty, L.: Picture-driven computing (January 2010), `http://web.mit.edu/newsoffice/2010/screen-shots-0120.html`
2. Lewis, J.P.: Fast Template Matching. In: Vision Interface 1995, pp. 120–123. Candian Image Processing and Pattern Recognition Society, Quebec City (1995)
3. Ouyang, W., Cham, W.K.: Fast algorithm for Walsh Hadamard transform on sliding windows. IEEE Transaction on Pattern Analysis and Machine Intelligence, 165–171 (2010)
4. Hofhauser, A., Steger, C., Navab, N.: Edge-Based Template Matching and Tracking for Perspectively Distorted Planar Objects. In: Bebis, G., et al. (eds.) ISVC 2008, Part I. LNCS, vol. 5358, pp. 35–44. Springer, Heidelberg (2008)
5. Lowe, D.G.: Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision (IJCV), 91–110 (2004)
6. Bay, H., Ess, A., Tuytelaars, T., Van Gool, L.: SURF: Speeded Up Robust Features. Computer Vision and Image Understanding (CVIU), 346–359 (2008)
7. Dalal, N., Triggs, B.: Histograms of Oriented Gradients for Human Detection. In: Conference on Computer Vision and Pattern Recognition, pp. 886–893 (2005)
8. Sun, Z., Liang, S.: Sketch retrieval and relevance feedback with biased SVM classification. Pattern Recognition Letters, 1733–1741 (2008)
9. Su, M.C., Hsio, T.H., Hsieh, Y.Z., Lin, S.C.: A neural-network-based sketch recognition system. In: International Symposium on Intelligent Signal Processing and Communications Systems (ISPACS), pp. 420–423 (2012)
10. Eitz, M., Hildebrand, K., Boubekeur, T., Alexa, M.: Sketch-Based Image Retrieval: Benchmark and Bag-of-Features Descriptors. Transactions on Visualization and Computer Graphics, 1624–1636 (2011)
11. Mourouzis, A., Leonidis, A., Foukarakis, M., Antona, M., Maglaveras, N.: A Novel Design Approach for Multi-device Adaptable User Interfaces: Concepts, Methods and Examples. In: Stephanidis, C. (ed.) Universal Access in HCI, Part I, HCII 2011. LNCS, vol. 6765, pp. 400–409. Springer, Heidelberg (2011)
12. Balagtas-Fernandez, F.T., Hussmann, H.: Model-Driven Development of Mobile Applications. In: International Conference on Automated Software Engineering (ASE 2008), pp. 509–512 (2008)
13. Kherraf, S., Lefebvre, E., Suryn, W.: Transformation From CIM to PIM Using Patterns and Archetypes. In: 19th Australian Conference on Software Engineering (ASWEC 2008), pp. 338–346 (2008)
14. Hammond, T.A.: LADDER: A Perceptually-based Language to Simplify Sketch Recognition User Interface Development, Massachusetts Institute of Technology, Doctor of Philosophy thesis (2007)
15. Diep, C.-K., Tran, Q.-N., Tran, M.-T.: Online model-driven IDE to design GUIs for cross-platform mobile applications. In: The 4th Symposium on Information and Communication Technology, pp. 294–300 (2013)