SoftwareX 12 (2020) 100599

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

ZWT: A new cross-platform graphical interface framework for Java applications

Simone Cirani [a], Marco Picone [b], Luca Veltri [c,*], Luca Zaccomer [c], Francesco Zanichelli [c]

[a] Caligoo Inc., USA
[b] Department of Sciences and Methods for Engineering, University of Modena and Reggio, Emilia, Italy
[c] Department of Engineering and Architecture, University of Parma, Italy

ARTICLE INFO

Article history:
Received 28 May 2020
Received in revised form 23 September 2020
Accepted 23 September 2020

Keywords:
Cross-platform
Java
Android
User Interface

ABSTRACT

The Java Programming Language revolutionized the world of software development in the last decades. Thanks to its portability, Java makes it possible to develop software that can run everywhere, in a truly cross-platform computing environment. Although running the same Java code anywhere works smoothly on major desktop and server platforms, this becomes much more complicated when different devices and platforms, such as smartphones or embedded systems, are taken into account. Furthermore, even if we consider devices that natively support the Java programming language, the same application may not run without re-writing part or the entire source code. This is mainly due to the existence of platform specific libraries for accessing input/output peripherals or system-specific features. In particular, the main limitation is usually associated to the different APIs that must be used for programming the Graphical User Interface (GUIs). In this paper, we present a novel framework that can be used by developers to write Java applications with portable GUIs that are truly platform-independent and thus can run on different systems such as PCs, Workstations, Android devices or mobile phones and embedded systems with Java MicroEdition (Java ME).

Code metadata

Current code version	v1.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX_2020_229
Code Ocean compute capsule	*None*
Legal Code License	Apache-2.0
Code versioning system used	Git
Software code languages, tools, and services used	Java, Android, Java ME
Compilation requirements, operating environments & dependencies	Java 8 or greater
If available Link to developer documentation/manual	https://github.com/zwt-sdk/zwt
Support email for questions	luca.veltri@unipr.it

1. Motivation and significance

"Write once, run anywhere (WORA)" was the original motto used by Sun Microsystems for describing the characteristics of the Java programming language. The possibility to implement an application independently from the platform(s) where it will actually run is particularly appealing when the programmer wants to develop a simple application without caring about the specific GUI widget toolkits available for the selected platform. On the other hand, it can be useful to develop an application that can be run, without changes, on different types of platforms (like PC, laptop, tablet, and smartphone), possibly without caring about the differences in the hardware and OSs.

We can say that an application is Cross-Platform (CP) when it satisfies the above development characteristics. In recent years, a great attention and focus have been paid to CP solutions in order to overcome the limitations and problems associated to managing different code bases for each mobile platform (e.g., Android and iOS). In [1–3], the authors analyze available frameworks, CP approaches, and existing Mobile application development platforms

* Corresponding author.
E-mail address: luca.veltri@unipr.it (L. Veltri).

https://doi.org/10.1016/j.softx.2020.100599

(MADPs), in order to understand the fragmentation level of the mobile landscape.

According to the Stack Overflow Developer Survey Results 2019 [4], the Java programming language is fifth among all used programming, scripting, and markup languages worldwide considering both professional and non-professional developers (41% among all respondents). Statistics and analysis provided by Jetbrains [5] and Baeldung [6] depict a prosperous Java development ecosystem where several frameworks and application servers are used to provide heterogeneous applications on different platform. Moreover, as confirmed by the Eclipse Foundation, the Java Programming language will also play a fundamental role in the Internet of Things ecosystem. Together with C/C++ and Python, Java is one of the top programming languages for the IoT and in particular it is leading the development of Edge Gateways and Cloud IoT applications [7]. In this scenario, the possibility to easily create CP Java graphical interfaces is really appealing not only considering traditional desktop and mobile platforms but also constrained devices and edge gateways that may take a significant advantage from a local UI.

Regarding Java applications, although the possibility to run the same Java code anywhere still succeeds for the desktop and server platforms in the spirit of WORA, this may not still be true when taking into account different devices, such as smartphones or embedded systems, even if the computing platform has native support for Java. Unfortunately, due to the existence of different and non-compatible libraries (e.g., related to input/output peripherals and/or system-specific features) an effective GUI portability is still an open issue forcing developers to rewrite parts or the entire code. In particular, the main problem is usually due to the existence of different platform-specific GUI widget toolkits that must be used for programming the UI, which strongly limits the portability of Java code.

In order to overcome this limitation, in this paper we present ZWT, a novel framework that allows developers to write graphical Java applications whose GUIs are truly platform-independent and can therefore run on different systems such as PCs, workstations, Android smartphones, or mobile phone and embedded systems with Java ME. With this work we focus our attention on the Java programming language and to the Android framework as its natural and native extension to the Mobile Computing ecosystem. However, this work is not meant to become yet another mobile cross-platform framework which, at the current stage, are out of the scope of our developments as well as other mobile platforms, such as iOS and Windows Phone. We would like to bring ZWT potentially to each platform capable to natively execute Java applications ranging from small constrained devices to high specs desktop computers.

In the context of existing Java graphical cross-platform applications and frameworks, JavaFX [8] represents the most relevant solution replacing Swing as the standard GUI library for Java SE. It is a rich and advanced platform for creating complex Java graphical applications that can run across a variety of devices and operating systems. ZWT is not meant to be a clone of JavaFX (or to replace Android UIs) but rather a lightweight alternative (in terms of footprint and dependencies) that can fit specific use cases related to the creation of "Micro User Interfaces" applicable to constrained environments (e.g., IoT Smart Objects) and that can be potentially re-used and integrated into existing and more complex frameworks (see the IoT controller example presented in Section 3). Furthermore, ZWT has been designed to provide a simple API (Application Programming Interface) similar to those available with Swing and AWT in order to support an easy porting of existing legacy applications into micro and re-usable independent graphical components.

2. Software description

In order to try to achieve the goals described in Section 1, we present the new ZWT (Zero-change Windows Toolkit) CP development framework. The main goal of the ZWT is to provide an easy-to-use solution for UI widget toolkit to be used in Java-based GUI applications that can run on almost any platform that supports the Java language, thus eliminating the gap introduced in codebases that depend on different UI libraries for different platforms (like AWT and Swing for Java SE, MIDP for Java ME, or Android API for Android).

Hereafter, the requirements that led the design and development of ZWT framework are summarized:

- *portability* - regardless of the UI library available on the underlying platform, an application developed using ZWT must be able to run on any platform without any code change. Our goal is not to completely duplicate the fully-featured APIs available on the specific platforms, but rather to have a simple and light core of UI APIs with a sufficient subset of features that allows the developer to create simple and cross-platform applications that can be executed in the true spirit of "WORA".
- *simplicity* - with ZWT developers do not have to learn a completely brand new UI framework with its own widget model and architecture, but rather they can use the same approach used by well-known libraries like AWT, Swing, or Android APIs. For this reason ZWT provides an API that resembles the same concepts of Swing and Android.
- *platform-agnostic* - the application does not have to be aware of the specific platform it is running on. That means that the code should not take into account the differences among different platforms and does not have to deal with those differences (i.e. without introducing `if-then-else` logic).
- *extensibility* - while ZWT must be as simple and light as possible, the API should be also easily extendible in order to let programmers personalize the graphical effects and/or to add new features.

2.1. Software architecture

Fig. 1 shows the architecture of ZWT. From the application point of view, ZWT provides a simple API that resembles well-known UI APIs like *Swing* of Java SE and the UI API of Android. This makes it very simple to implement cross-platform applications without binding them to the actual UI libraries of all platforms.

The implementation of ZWT is formed by two layers. The upper layer (i.e. ZWT "independent" objects and interfaces in Fig. 1) is composed by all ZWT interfaces and all classes that are independent by underlying platforms in order to abstract them: their code is completely CP, and based on a restricted number of classes forming the underlying layer. The lower layer (i.e. the ZWT 'primitive' objects in Fig. 1) is where the ZWT implementation is concretely bound to the specific platform. Despite this layer still provides the same API to the upper layer and to the user, its internal implementation differs from the specific platforms and is based on APIs available on those platforms. Currently, three concrete binding implementations are available: (i) Java SE, using the Java Swing API, (ii) Java ME, using the Mobile Information Device Profile (MIDP) of Java ME, and (iii) Android, using the standard UI API of Android. This layer actually makes ZWT capable of running on different operating systems and hardware while using the same high-level API.

Note that the lower layer of ZWT contains only the core components of the UI, which are then used by all other components. This lets developers to easily extend those components or to create new ones without having to be aware of the underlying platform(s).

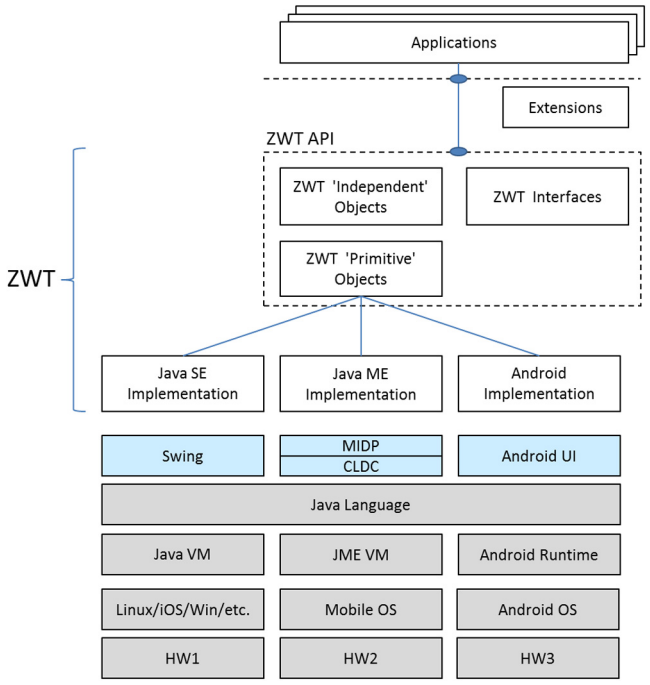

Fig. 1. ZWT architecture.

2.2. Software functionalities

From the developer's perspective, ZWT is formed by a basic set of standard interfaces and classes, contained in the `it.unipr.netsec.zwt` package, which provides all basic components and functions for building a UI. Other additional and optional packages contain extensions, integrations, and effects.

The `it.unipr.netsec.zwt` package includes panels, labels, buttons, menus, event callbacks, and drawing methods. The relationships among the entities of this set of basic components are shown in the class-diagram reported in Fig. 2.

Some extension packages that are already included are `layout`, `border`, `floor`, `menu`, and `keyboard`.

2.3. Sample code snippets analysis

In order to show the simplicity of running the same application on different platforms, hereafter we report some sample codes used for running an app on Java SE, Java ME, and Android.

Since the ZWT library resembles other well-known GUI libraries, writing an app with CP UI is very simple. Hereafter, we show a snippet of code of a basic *Hello* application that displays a label "Hello world" and an "Ok" button for exiting:

```
1  import it.unipr.netsec.zwt.*;
2  import it.unipr.netsec.zwt.layout.ZwtBorderLayout;
3
4  public class Hello {
5
6   public Hello(ZwtFrame frame) {
7    frame.setLayout(new ZwtBorderLayout());
8    ZwtLabel label=new ZwtLabel("Hello world");
9    label.setColor(ZwtColor.WHITE);
10   label.setAlignment(ZwtLabel.ALIGN_HCENTER);
11   frame.addComponent(label,ZwtBorderLayout.CENTER);
12   ZwtButtonListener listener=new ZwtButtonListener()
         {
13    @Override
14    public void onButtonPushed(ZwtButton arg0) {
15     System.exit(0);
16    }
17   };
18   ZwtButton button=new ZwtButton("Ok",ZwtKeyboard.
         KEY_SELECT,listener);
19   frame.addComponent(button,ZwtBorderLayout.SOUTH);
20  }
21 }
```

In this example, the constructor takes a `ZwtFrame` argument and uses it to draw the UI (in this case only a label and a button). As we can see, the code has no direct binding to any particular UI widget toolkit and is therefore platform-independent.

In order to run the example on Java SE, all we need is a just a class with a main method that creates a `ZwtFrame` based on a Java *Swing* `JFrame` and passes it to the constructor of `Hello`, so that the dependency on Swing is injected. Here is an example of the main class:

```
1  import javax.swing.JFrame;
2
3  import it.unipr.netsec.zwt.ZwtFrame;
4
5  public class HelloMain {
6
7   public static void main(String[] args) {
8    JFrame jframe=new JFrame();
9    ZwtFrame frame=new ZwtFrame(jframe,200,100);
10   new Hello(frame);
11  }
12 }
```

The same app can be easily built for an Android platform by simply creating a main Activity with an `ImageView` and no status bar.

Here is an example of code for the Android Activity:

```
1  import android.graphics.Point;
2  import android.support.v7.app.AppCompatActivity;
3  import android.os.Bundle;
4  import android.view.Display;
5  import android.widget.ImageView;
6
7  import it.unipr.netsec.zwt.ZwtFrame;
8
9  public class HelloMainActivity extends
       AppCompatActivity {
10
11  @Override
12  protected void onCreate(Bundle savedInstanceState) {
13   super.onCreate(savedInstanceState);
14   setContentView(R.layout.activity_hello_main);
15
16   int statusBarHeight = 0;
17   int resourceId = getResources().getIdentifier("
         status_bar_height", "dimen", "android");
18   if (resourceId > 0) statusBarHeight = getResources
         ().getDimensionPixelSize(resourceId);
19   Display display = getWindowManager().
         getDefaultDisplay();
20   Point size = new Point();
21   display.getSize(size);
22   int width = size.x;
23   int height = size.y - statusBarHeight;
24
25   ImageView imageView = findViewById(R.id.myImageView
         );
26
27   ZwtFrame frame = new ZwtFrame(imageView, width,
         height);
28   new Hello(frame);
29  }
30 }
```

Note that the Activity just creates a `ZwtFrame` based on an `ImageView` and uses it to invoke the constructor of `Hello`.

Finally, in case of Java ME, the app can be run easily by just writing a simple Java ME Midlet that creates a `ZwtFrame` and passes it as an argument to the `Hello` constructor, as follows:

```
1  import it.unipr.netsec.zwt.ZwtFrame;
2
3  import javax.microedition.midlet.MIDlet;
4
```

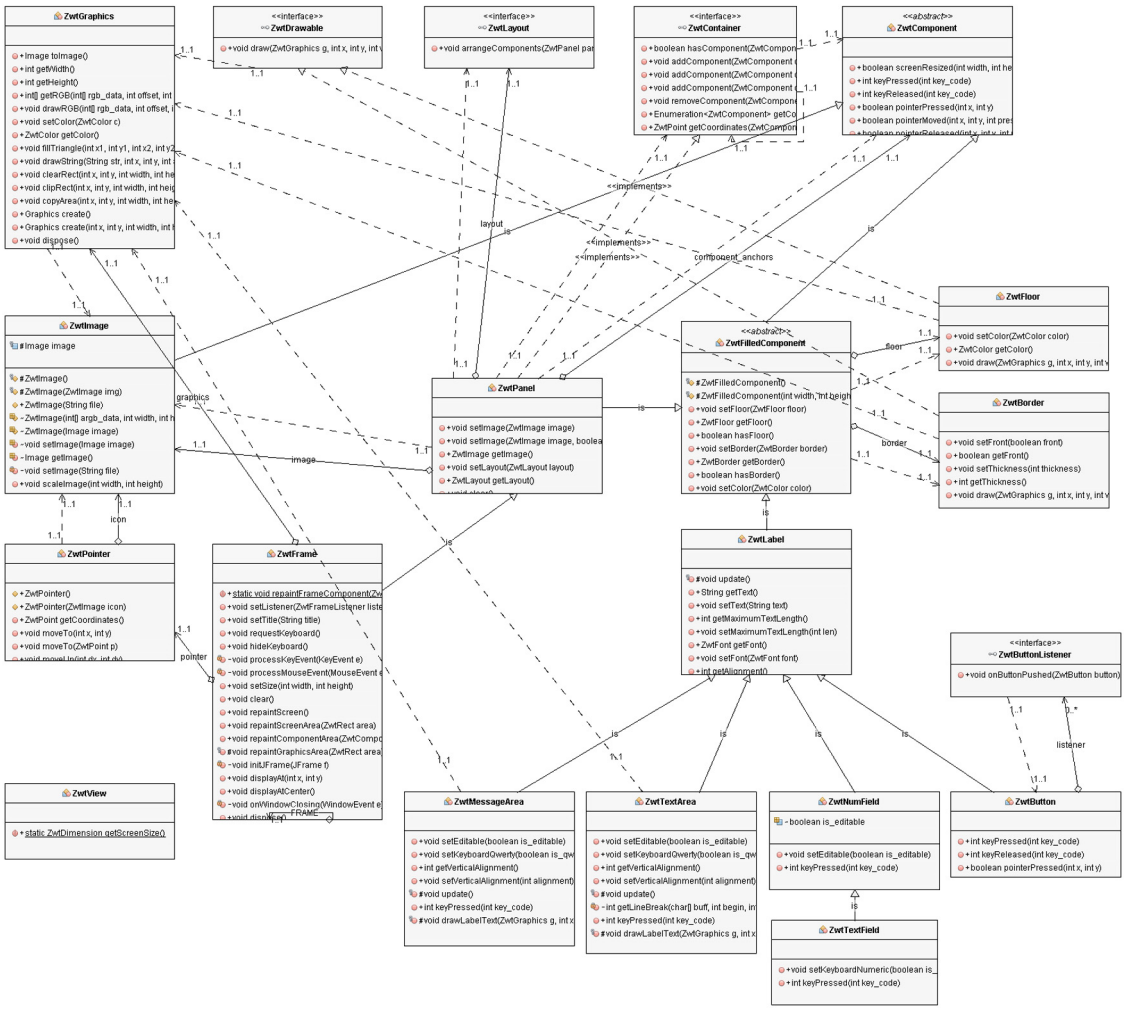

Fig. 2. ZWT core objects.

```
5  public class HelloMIDlet extends MIDlet {
6
7   public HelloMIDlet() {
8    ZwtFrame zf=new ZwtFrame(this);
9    new Hello(zf);
10   zf.repaintScreen();
11  }
12
13  public void startApp() {}
14
15  public void pauseApp() {}
16
17  public void destroyApp(boolean unconditional) {}
18 }
```

This application is very simple, but even more complex applications with richer GUIs (like the examples shown in Section 3) require the same code shown above to actually perform the dependency injection in for the three different platforms.

3. Illustrative examples

In order to illustrate the potential of the ZWT framework, three sample applications are presented. The first is a simple cross-platform calculator with Reverse Polish Notation. The calculator code is available in the `test.calc` package and, as expected, is completely independent from the underlying platforms. As a result, the application can run unmodified on either an Android phone, a Java ME enabled phone, or a PC with Java SE. It is responsive and adapts correctly to the displays of different platforms, rotating to landscape-mode when the phone supports display rotations.

As stated in Section 2, the framework is easy to extend: new components or graphical effects can be added without being aware of the underlying platform and knowing the native API. Any additional graphical component can be developed on top of core ZWT objects.

Some simple graphical effects like transparency and roundness are already available in ZWT (check packages `floor` and `border`) and used in this demo application. An example of the calculator with transparency effects running on a mobile phone with Java ME is shown in Fig. 3(a).

The application has the following two peculiarities. On the one hand, it provides its own keyboard and consequently a platform software keyboard (e.g. in case of smartphones with touchscreen) is not required while a native keyboard can be still used for example in case of PCs and some mobile phones (like the one shown in Fig. 3(a)). On the other hand, the application's UI is always completely framed in the display and scrolling is not required.

In order to exploit these last two functionalities not included in the calculator example, a second example is hereafter illustrated.

An Instant Messaging (IM) cross-platform application and the corresponding User Agent (UA) have been developed.

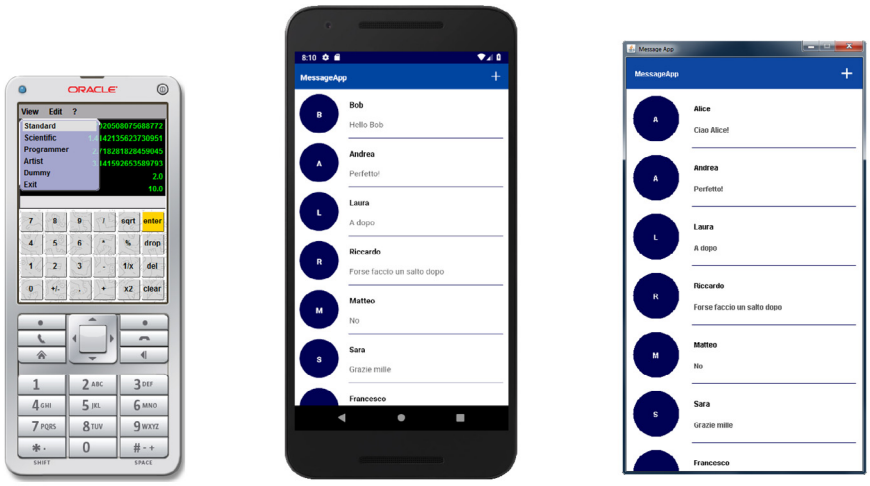

Fig. 3. Examples: (a) calculator with transparency effect, (b) instant messaging app on Android phone, (c) same app running on PC.

Since this is an example application, only basic IM functionalities have been implemented: (i) it presents a login interface to the user where the user can enter his/her username; (ii) it registers the UA (and the username) to a remote server that is used to dispatch incoming and outgoing messages among users; (iii) it adds new contacts; (iv) it shows a buddy list to the user, displaying the contacts and a preview of the last exchanged message; (v) displays the complete chat for each selected contact; and (vi) it sends and receives messages.

The IETF standard Session Initiation Protocol (SIP) [9] has been selected as IM signaling protocol. Support for SIP operations (register/unregister/send/receive) is provided by the mjSIP library [10], which is an open source cross-platform SIP implementation that can run on Java SE, Android, and Java ME.

In order to manage the user's contact registration and message relaying, we used the mjSIP Session Border Controller (SBC) with Registrar and Proxy functions.

The resulting sample application is open-source. In Fig. 3 the same IM app UI is shown when running on an Android phone (b) and PC (c).

The third example is a UI for controlling IoT devices. This scenario is particularly relevant: as several IoT products are being brought to the market, a unified and consistent codebase for user-facing applications may be convenient to distribute products that work with connected objects on different devices. Starting from the main panel, as shown in Fig. 4(a) and (d) running on a PC and on an Android phone respectively, it is possible to select a device. The corresponding control panel is then displayed. Examples are shown in Fig. 4(b) and (c) for a PC, (e) for an Android phone, and (f) for a Java ME phone. The UI can be fully customized and different graphical components can be displayed. Moreover, the UI is fully independent from the underlying platform and it runs transparently on either a PC, Android, or a Java embedded platform.

All three illustrated demo applications are available on the ZWT repository [11].

4. Impact

ZWT represents an innovative point of view for developing Java Graphical Cross Platform applications. The consolidated widespread adoption of the Java programming language combined with the availability of experienced developers and the huge market share of Android are the perfect conditions to make Java an ecosystem that can overcome the limitations of "write once and redesign (the UI) everywhere". The proposed framework aims at allowing developers to write Java applications whose GUIs are truly independent from the underlying layers and the running platforms. This approach has a direct impact on the reuse of code and Graphical User Interface and at the same time it protects the application from platform-specific constraints or hard changes.

Actually, the ZWT framework allows developers to create graphical CP applications that can be executed without any change or adaptation on Java Micro Edition (JME), Standard Edition (JSE) and Android with a single code base. This approach simplifies the transition of existing applications from Java ME and SE to Android and to new classes of devices that were not available (and even imaginable) at the time of their design. This migration will be easier, cost effective, and furthermore will be less influenced by a skill gap. Java developers can start working immediately on the software without the need of learning a new platform and keeping the focus on Java programming language both for the core and the User Interface.

Sometimes, and in particular for the Mobile ecosystem (as illustrated and analyzed in [12]), this skill gap brought companies to develop workarounds to support some sort of portability, such as web-based cross platforms solutions. However this technological debt eventually requires to entirely re-write the application to integrate updates and overcome incompatibility issues.

5. Conclusions

The Java cross-platform graphical interface framework presented in this paper is a novel, scalable, and flexible solution targeting different Java development application scenarios. It can be used to develop cross-platform graphical applications for the Java Standard and Micro Editions and the same code can be also executed and integrated in Android mobile apps without any changes or significant readaptation of the codebase. Furthermore, the possibility to design and write an application's graphical user interface and reuse it on different platforms without changes is appealing also for dynamic user interface generation scenarios, both for form-oriented applications [13] or Internet of Things scenarios, where the UI can be dynamically generated according to the Smart Object type and functionalities [14].

In this article, we have described how this framework has been designed and implemented and how it is possible to setup and create a graphical cross-platform Java application. Three sample applications (with increasing complexity) have been presented

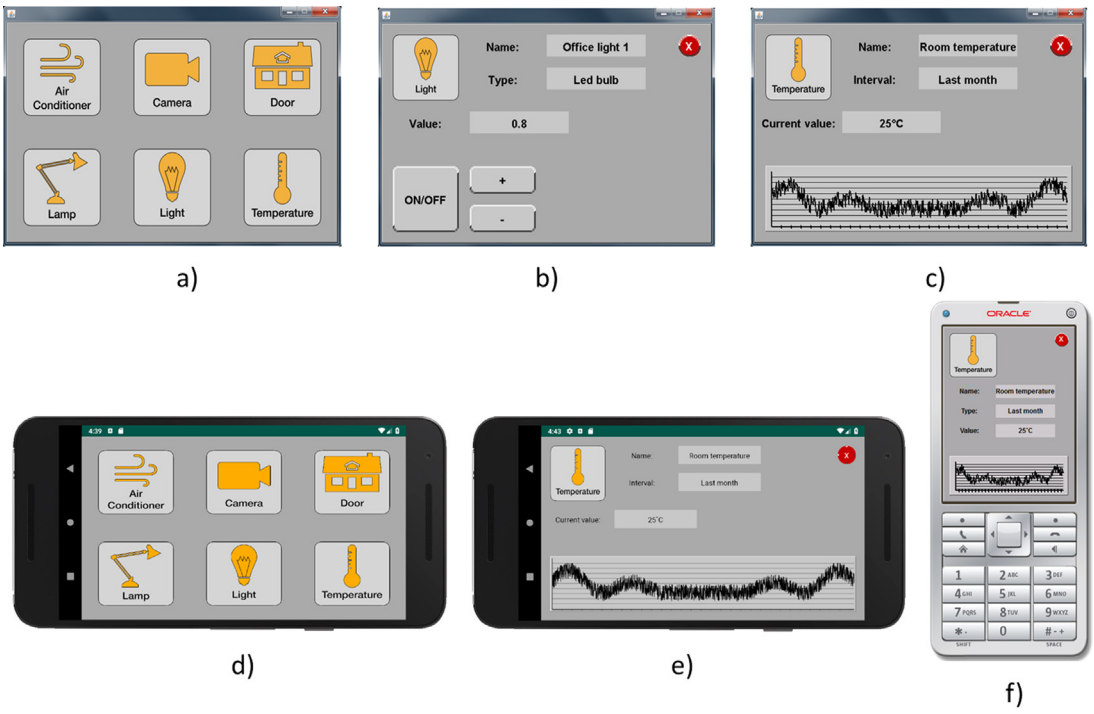

Fig. 4. Examples: IoT control app running on a PC, Android phone, and Java ME phone: (a) The main control panel on Java SE, (b) Device control panel on Java SE, (c) Device monitoring panel on Java SE, (d) The main control panel on Android, (e) Device monitoring panel on Android, (f) Device monitoring panel on JME with vertical display.

and tested on Java SE, Java ME, and Android. A more detailed description of the software and all its possible usages can be found on the official website and repository. ZWT is not intended to become a cross-language UI toolkit but is limited to Java-enabled applications only, as discussed in Section 1. Currently, the most relevant missing features compared to other frameworks are mainly related to the availability of richful UI components, such as in-app embedded web views and media player controllers, and the possibility to provide styling to applications through external configurations, such as using CSS. ZWT aims to be an active and ongoing project and we are planning to improve the list of supported features and tools together with a extended set of examples and demo applications for a wide range of solutions also including embedded and constrained devices.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Smutný P. Mobile development tools and cross-platform solutions. In: Proceedings of the 13th international Carpathian control conference (ICCC). 2012, p. 653–6. http://dx.doi.org/10.1109/CarpathianCC.2012.6228727.
[2] Pinto CM, Coutinho C. From native to cross-platform hybrid development. In: 2018 international conference on intelligent systems (IS). 2018, p. 669–76. http://dx.doi.org/10.1109/IS.2018.8710545.
[3] Biørn-Hansen A, Grønli T-M, Ghinea G. A survey and taxonomy of core concepts and research challenges in cross-platform mobile development. ACM Comput Surv 51(5). http://dx.doi.org/10.1145/3241739.
[4] Stack overflow developer survey results. 2019, URL https://insights.stackoverflow.com/survey/2019. [Last Accessed 08 May 2020].
[5] Java development ecosystem. 2019, https://www.jetbrains.com/lp/devecosystem-2019/java/. [Last Accessed 08 May 2020].
[6] Baeldung - The state of java in 2019. 2019, URL https://www.baeldung.com/java-in-2019. [Last Accessed 08 May 2020].
[7] Eclipse - IoT developer survey 2019 results. 2020, URL https://iot.eclipse.org/community/resources/iot-surveys/. [Last Accessed 08 May 2020].
[8] JavaFX. 2020, URL https://openjfx.io/. [Last Accessed 08 May 2020].
[9] Schooler E, Rosenberg J, Schulzrinne H, Johnston A, Camarillo G, Peterson J, et al. SIP: Session initiation protocol, RFC 3261. 2002, http://dx.doi.org/10.17487/RFC3261, URL https://rfc-editor.org/rfc/rfc3261.txt.
[10] Veltri L, Fadda C. An open-source platform for ip telephony services. In: 2007 15th international conference on software, telecommunications and computer networks. 2007, p. 1–5. http://dx.doi.org/10.1109/SOFTCOM.2007.4446107.
[11] ZWT Repository. 2020, URL https://github.com/zwt-sdk. [Last Accessed 18 September 2020].
[12] Daradkeh MK, Sabbahein HAS. Factors influencing the adoption of mobile application development platforms: A qualitative content analysis of developers' online reviews. Int J Enterp Inf Syst (IJEIS) 2019;15(4):43–59.
[13] Galizia A, Zereik G, Roverelli L, Danovaro E, Clematis A, D'Agostino D. JsonGUI - A module for the dynamic generation of form-based web interfaces. SoftwareX 2019;9:28–34. http://dx.doi.org/10.1016/j.softx.2018.11.007.
[14] Belli L, Cirani S, Gorrieri A, Picone M. A novel smart object-driven ui generation approach for mobile devices in the internet of things. In: Proceedings of the 1st international workshop on experiences with the design and implementation of smart objects. 2015, p. 1–6. http://dx.doi.org/10.1145/2797044.2797046.