Вестник Сыктывкарского университета.
Серия 1: Математика. Механика. Информатика. 2022.
Выпуск 1 (42)
Bulletin of Syktyvkar University.
Series 1: Mathematics. Mechanics. Informatics. 2022; 1 (42)

ИНФОРМАТИКА

Научная статья

УДК 539.3
https://doi.org/10.34130/1992-2752_2022_1_61

Development of XML-based Markup Language

Vadim A. Melnikov[1], **Andrey V. Yermolenko**[2]
[1,2]Pitirim Sorokin Syktyvkar State University, [1]ea74@list.ru

Abstract. Modern approaches in the field of software development assume not only the functionality of the product being developed, but also the convenience, clarity and familiarity of the interfaces. Today, the developed software can be used on various devices, with different configurations, and users may also need a different language to work with the software. To address the issue of universality in the field of 2D games, the approach used in the development of the user interface for the Sad Lion Engine is proposed. Within the framework of this approach, it is supposed to use the markup language Sad Lion Markup Language, the description and use of which is given in the article.

Keywords: user interfaces, C++, mobile development, markup languages

For citation: Melnikov V. A., Yermolenko A. V. Development of XML-based Markup Language. *Bulletin of Syktyvkar University, Series 1: Mathematics. Mechanics. Informatics*, 2022, No. 1 (42), pp. 61–73. https://doi.org/10.34130/1992-2752_2022_1_61

Язык разметки SadLion Markup Language

Вадим Андреевич Мельников[1], **Андрей Васильевич Ермоленко**[2]

[1,2] Сыктывкарский государственный университет имени Питирима Сорокина

Аннотация. Современные подходы в области разработки программного обеспечения (ПО) предполагают не только функциональность разрабатываемого продукта, но и удобство, понятность и привычность интерфейсов. На сегодняшний день разрабатываемое ПО может использоваться на различных устройствах, с различными конфигурациями, а также пользователям может быть необходим другой язык для работы с ПО. Для решения вопроса универсальности в области 2D-игр предлагается подход, использованный при разработке пользовательского интерфейса игрового движка Sad Lion Engine. В рамках данного подхода предполагается использовать язык разметки Sad Lion Markup Language, описание и использование которого приведено в статье.

Ключевые слова: пользовательский интерфейс, Си++, разработка мобильных приложений, языки разметки

Для цитирования: Melnikov V. A., Yermolenko A. V. Development of XML-based Markup Language // *Вестник Сыктывкарского университета. Сер. 1: Математика. Механика. Информатика.* 2022. Вып. 1 (42). С. 61–73. https://doi.org/10.34130/1992-2752_2022_1_61

Introduction

The user interface is just important as a program code. According to the developers' estimates, the end user first of all pays attention to the program interface, and not to its functionality. Therefore, the special attention is always given to the questions of interface.

After the first version of Unix appeared in 1969, programs were designed to work in text-based terminals, and a text-based interface was the norm. In the mid-80s, graphical user interfaces began to appear (hereinafter GUI), the development of which from the early 70s was carried out by Xerox Palo Alto Research Center. This research center influenced the development of the Apple Macintosh interface, and through it, the Microsoft Windows interfaces [1].

Today it is necessary not only to create a user interface for the program, but also to design it taking into account internationalization, because most modern programs are localized into many different languages. And when developing for mobile devices, it is also necessary to take into account the existence on the market of devices with different screen ratios, since in addition to the already familiar 9:16 ratio, which is widespread among phones, and 3:4, used for tablets, there is also a 9:19.5 ratio for smartphones Apple iPhone X[1].

Various methods of layout development are used to create user interfaces on mobile devices. One of the easiest ways is to use the built-in browser engine (usually such a component is called WebView) and write code using JavaScript and HTML, which allows Apache Cordova [2]. But this approach will lead to the fact that the application will be initially slow, annoying users. The second way is to create native Android [3] and iOS [4] apps using standard layout tools or cross-platform Xamarin [5]. Both ways have similarities and are all very similar to the older Windows Presentation Foundation (WPF) technology [6]. At first glance, it may seem that the technology that will be described below reinvents HTML and JS, but it should be noted that for using HTML and JS on rather weak (compared to PCs based on x86 architecture) mobile platforms, it leads to an increase in power consumption and a decrease in performance applications. Nikolay Armonik in his report "How we made the biggest game on Defold in 1 year" at the DevGAMM Moscow 2018 conference gives figures on the performance of their application: 20-40 % of the frame time is taken by the execution of Lua scripts, Lua is interpreted language like JavaScript and they are comparable in performance. Ditcing interpreted languages and writing all the logic in C ++ greatly improves application performance.

The native development approach forces developers to maintain several completely incompatible codebases — one in Java or Kotlin for Android, and one in Objective-C or Swift for iOS in the case of mobile applications. If you're developing for Windows, you'll need to add the UWP. Part of the problem is solved by using Xamarin Forms [5], which allow you to generalize most of the code and only in special cases write code using Xamarin Native.

The Flutter [7] technology stands apart, in which the native elements of the user interface are not used, but the elements are drawn immediately on

[1]According to sites: https://deviceatlas.com/blog/most-used-smartphone-screen-resolutions, https://www.apple.com/ru/newsroom/2017/09/the-future-is-here-iphone-x/

the screen using Skia[2] – the Google vector graphics library. The approach is quite similar to designing interfaces in games. Flutter uses the Dart language for programming, which is focused on Web applications and UI, just like JavaScript. Dart is not as powerful as C++, and in games, performance is critical.

The given review of the modern approach to the creation of interfaces shows that the demanded task is to create universal, easily programmable interfaces that adjust to various devices and screens, with different aspect ratios. To solve this problem, it is proposed to use the Sad Lion Engine (SLE) game engine (SLE) [8] user interface, created with the help of C++, based on the developed markup language for maximum application performance. Accordingly, the purpose of the article is to describe the capabilities of the developed interface of the SLE game engine. It should be noted that there is a powerful framework Qt [9] for developing applications for PCs and mobile platforms, but it was not initially considered due to the prohibitive cost for a small development studio.

This approach is applicable both to business applications and to any games (2D, 3D, AR, VR), since even in 3D games, the interface remains a 2D texture placed in space.

A review of publications in scientific and technical literature proves that, despite the rapid development of programming languages and environments, mobile game developers face a fairly large number of various difficulties when creating game applications and interfaces. For example, general problems of game design, including the development of mobile applications, are raised in [10; 11]. Developers are considering different approaches to solving problems related to interfaces. In [12], the authors present the XVGDL language for describing games, based on the extensible markup language XML, while emphasizing the universality of XVGDL, describing the syntax and components of XVGDL, and examples of their use. In the study in [13], the authors present an approach for the automated creation of classic multiplayer 2D games based on Android, based on the concept of model-based engineering. The development of lightweight performance game engines also remains a popular trend. However, this raises questions of their development, the introduction of new technologies and the functionality provided by modern and promising devices [14].

[2]https://skia.org/ – Skia vector graphics library site.

First version of markup language made for Unity3D

The first version of Sad Lion Markup Language (SLML) ran entirely within Unity3D. The creation of the interface was based on the component system [10] and a set of components of the Unity3D engine itself. Application lifecycle was also based on Unity3D [11]. The emergence of SLML in Unity3D is justified by the complexity of creating a high-quality adaptive interface using standard engine tools. It is quite difficult to align UI elements so that they look good on different screen ratios.

The emergence of SLML and SLE began with a solution to the problem of large executable file sizes. When using sprites, Unity3D embed them in an unpacked executable file, which led to a large increase in the size of the application. With further development, it was revealed that adding new objects also leads to a strong increase in the size of the executable file. To solve this problem with sprites, a special tool was made for creating atlases in PNG files. Solving the problem with objects required creating our own system for creating objects in the scene, bypassing Unity3D, and this was the reason for the appearance of SLML.

SLML does not create objects in advance, all descriptions are stored in XML files, objects are set on the fly when the game starts. This approach made it possible to avoid embedding the code of objects in the binary code, instead, they are generated according to the description from the XML file as needed.

In SLE, scaling occurs after all objects are created. In Unity3D, the game is executed within the Awake-Start-Update cycle, which leads to the following problem: it is necessary to create objects in the framework, in which there is no possibility to create before the cycle starts. Also, adjusting the width-height using anchors in Unity3D is not very convenient.

The difficulties described above indicated the need to develop our own SLE core.

Implementation within SLE

SLML is based on XML [17]. As part of SLE, the problems related to the life cycle of an application in Unity3D were solved, the life cycle of SLE involves the complete creation of a component tree before the first call to Awake of any of the components specified in the XML file. Using the Skia library allowed adding a number of new functions to the user interface system.

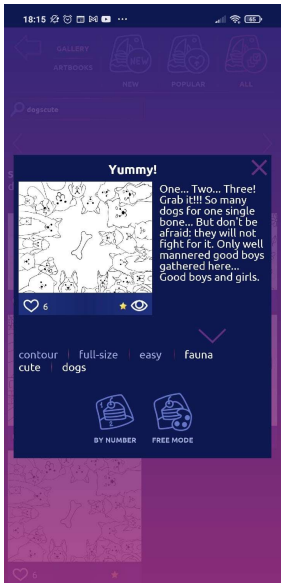

Fig. 1. Screen with aspect ratio 9:16

The main idea of SLML is to use an adaptive layout, in which several (usually three) layouts are initially set up and then the user interface is interpolated based on one of the existing layouts. If there is a relationship for which high-quality interpolation does not work, then it can be implemented separately without affecting the engine itself.

Let's show the advantages of SLE on the example of the Big Coloring Book[3] project, using a form layout made for one and two ratios. In the current version of SLML, additional layouts are obtained using variables stored in a JSON file and substituted during parsing of XML files.

Let's take an aspect ratio close to 9:16 (Xiaomi Redmi 4X) as a basis, the result is shown in Fig. 1. Fig. 2 shows how the interface will look in a ratio close to 3: 4 (DEXP E170), obtained by proportionally scaling the interface created for a ratio of 9:16. Immediately, a strong violation of proportions is noticeable, the pop-up window is much larger in width than in height, and the elements also begin to run over each other.

Fig. 3 shows the result obtained using a separate 3:4 ratio. This approach allowed us to obtain a better result for each aspect ratio, the proportions of elements and the distances between them are preserved. According to professional designers, in this case, with the help of optical alignment, the image becomes visually balanced and harmonious for the user to perceive

[3]App in the Google Play store — https://play.google.com/store/apps/details?id=sadlion.games.bigcoloringbook.

Fig. 2. Screen with aspect ratio 3:4, obtained by scaling from layout with aspect ratio 9:16

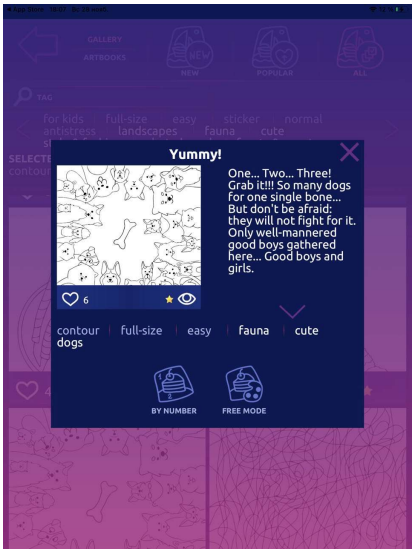

Fig. 3. Screen with aspect ratio 3:4

at an intuitive level. The block with the text has enough free space around it and is related from the point of view of visual ergonomics. If the ratio of the sides is not from the list of those implemented in advance, then proportional scaling from the closest ratio will occur. This allows you to take into account the specifics of the screen ratio, large width and vice versa, height, and position the interface elements more accurately.

Within the SLM, there are several main tags that provide basic functionality:

- *Form* is the root tag for layout file;
- *Row* is a tag of row which takes all width and given height of parent, rows are placed one after another;
- *Col* is a tag of column which takes the given width and all height of parent, columns are placed one after another;
- *Layer* is a tag of layer, which takes all space of parent and placed in point (0, 0) regarding the parent, layers overlap each other;
- *Button* is a tag of button, which gives opportunity to handle user gestures like taps and moves of finger;
- *Image* is a raster sprite;
- *Text* is a one-line text;

Let's now look at the most important attributes for tags:

- *width* – in the case of the Form tag, it can only have an absolute value in pixels, it sets the screen width, in the case of the rest Col can be expressed as a percentage and sets the column width, in other cases it is ignored;
- *height* – in the case of the Form tag, it can only have an absolute value in pixels, sets the screen height, in the case of the rest, Row can be expressed as a percentage and sets the row height, in other cases it is ignored;
- *name* – sets the string name of the object, by which it can be found from C++ code;
- *aspectRatioFitter* – the way to control aspect ratio of object, now only FitInParent is supported;
- *components* – attribute, which contains names of components, which must be connected to the object, in C# version this was made with the reflection, in C++ this is done by register of object instantiators;

- *text* – string attribute which sets the value to tag Text;
- *group onUp, onDown, onClick* – attributes of button events, which store names of events handlers. This functionality is realized with the register of event handlers in each form;
- *sprite* – string constants in format "atlas/index", where atlas is name of the atlas and index is number of the sprite in the atlas.

This is a list of the main attributes, the rest of the attributes only provide visual effects such as gradients, frames, etc. They are of interest only for documentation purposes. Below is an example source code snippet for the Big Coloring Book project:

```
<Row height="24.5">
  <Col color="255/0/255/0">
  <Text name="txtMyWorks"
    font-size="28"
    font="Ubuntu-Bold"
    color="+/lightBlue"/>
  </Col>
</Row>
```

Conclusion

After switching from Unity3D to Xamarin C# [8], the functionality of the engine and SLML was increased, the lifecycle was adapted to the requirements of SLE and SLML, and the loading speed of applications increased greatly.

After switching to C++, performance has increased significantly, but the flexibility and ease of C# development has been lost. In C++, there is no reflection and all components and handlers have to be registered manually, and the Android Studio IDE also leaves much to be desired in terms of performance compared to Visual Studio from Microsoft: it takes a long time to compile, the debugger is unstable (while working with a smartphone xiaomi redmi 4, the debugger is disabled at the point stops), slow hints on possible function names (and with an application size of 20 thousand lines, it is impossible to keep everything in memory), but Xamarin C# did not have sufficient stability.

The SLML language is in constant development and continues to change along with SLE, from the last changes we note the functions concerning the

positioning of objects and the displacement of adjacent ones when the height of one of the objects changes.

Список источников

1. **Rago A. S., Stevens W. R.** Advanced programming in the UNIX environment. Addison-Wesley Professional, 2013. 1032 p.

2. **Camden K. R.** Apache Cordova In Action. Shelter Island: Manning, 2016. 230 p.

3. **Thornsby J.** Android UI design. Birmingham: Packt Publishing, 2016. 356 p.

4. **Bennett G., Kaczmarek S., Lees B.** Swidt 4 for Absolute Beginners. Phoenix: Apress, 2018. 317 p.

5. **Petzold C.** Cross-platform C# programming for iOS, Android and Windows. Redmond, Washington: Microsoft Press, 2013. 1161 p.

6. **Petzold C.** Applications = Code + Markup. A Guide To the Microsoft Windows Presentation Foundation. Redmond: Microsoft Press, 2006. 1002 p.

7. **Windmill E.** Flutter in Action. Shelter Island: Manning, 2020. 368 p.

8. **Мельников В.А.** Процесс разработки движка для 2D-игр и интерфейсов Sad Lion Engine // *Вестник Сыктывкарского университета. Сер.1: Математика. Механика. Информатика*, 2019. Вып. 4 (33). С. 21–37.

9. **Dogsa T., Meolic R.** A C++ App for Demonstration of Sorting Algorithms on Mobile Platforms // *International Journal of Interactive Mobile Technologies*, 2014 Vol. 8, No 1. URL: https://www.online-journals.org/index.php/i-jim/article/view/3464/2940 (дата обращения: 17.07.2020).

10. **Cao J., Cao Y.** Application of human computer interaction interface in game design // *Communications in Computer and Information Science*, 2017, 714, pp. 103–108. DOI: 10.1007/978-3-319-58753-0_16.

11. **Zaina L. A. M., Fortes R. P. M., Casadei V., Nozaki L. S., Paiva D. M. B.** Preventing accessibility barriers: Guidelines for using user interface design patterns in mobile applications // *Journal of Systems and Software*, 2022, vol. 186. DOI: 10.1016/j.jss.2021.111213.

12. **Quinones, J. R., Fernandez-Leiva, A. J.** XML-Based Video Game Description Language // *IEEE Access*, 2020, vol. 8, pp. 4679-4692. DOI: 10.1109/ACCESS.2019.2962969.

13. **Derakhshandi M., Kolahdouz-Rahimi S., Troya J., Lano K.** A model-driven framework for developing android-based classic multiplayer 2D board games // *Automated Software Engineering*, 2021, 28 (2). DOI: 10.1007/s10515-021-00282-1.

14. **Park, H. C., Baek, N.** Design of SelfEngine: A Lightweight Game Engine // *Lecture Notes in Electrical Engineering*, 2020, vol. 621, pp. 223–227. DOI: 10.1007/978-981-15-1465-4_23.

15. **West M.** Evolve your hierarchy [Online] // Cowboy Programming. URL: http://cowboyprogramming.com/2007/01/05/evolve-your-heirachy/ (дата обращения: 22.08.2020).

16. **Hocking J.** Unity in action: Multiplatform game development in C#. Manning, 2018. 400 p.

17. **Lisovsky, K.Y.** XML Applications Development in Scheme // *Programming and Computer Software 28*, pp. 197–206 (2002). https://doi.org/10.1023/A:1016319000374

References

1. **Rago A. S., Stevens W. R.** *Advanced programming in the UNIX environment.* Addison-Wesley Professional, 2013. 1032 p.

2. **Camden K. R.** *Apache Cordova In Action.* Shelter Island: Manning, 2016. 230 p.

3. **Thornsby J.** *Android UI design.* Birmingham: Packt Publishing, 2016. 356 p.

4. **Bennett G., Kaczmarek S., Lees B.** *Swidt 4 for Absolute Beginners.* Phoenix: Apress, 2018. 317 p.

5. **Petzold C.** *Cross-platform C# programming for iOS, Android and Windows.* Redmond, Washington: Microsoft Press, 2013. 1161 p.

6. **Petzold C.** *Applications = Code + Markup. A Guide To the Microsoft Windows Presentation Foundation.* Redmond: Microsoft Press, 2006. 1002 p.

7. **Windmill E.** *Flutter in Action.* Shelter Island: Manning, 2020. 368 p.

8. **Melnikov V. A.** Development Process of game engine core for 2D games and interfaces Sad Lion Engine. *Vestnik Syktyvkarskogo universiteta. Ser.1: Matematika. Mexanika. Informatika* [Bulletin of Syktyvkar University. Series 1: Mathematics. Mechanics. Informatics], 2019, 4 (33), pp. 21–37. (In Russ.)

9. **Dogsa T., Meolic R.** A C++ App for Demonstration of Sorting Algorithms on Mobile Platforms. *International Journal of Interactive Mobile Technologies.*, 2014. Vol. 8, No 1. Available: https://www.online-journals.org/index.php/i-jim/article/view/3464/2940 (accessed: 17.07.2020).

10. **Cao J., Cao Y.** Application of human computer interaction interface in game design. *Communications in Computer and Information Science*, 2017, 714, pp. 103–108. DOI: 10.1007/978-3-319-58753-0_16.

11. **Zaina L. A. M., Fortes R. P. M., Casadei V., Nozaki L. S., Paiva D. M. B.** Preventing accessibility barriers: Guidelines for using user interface design patterns in mobile applications. *Journal of Systems and Software*, 2022, vol. 186. DOI: 10.1016/j.jss.2021.111213.

12. **Quinones, J. R., Fernandez-Leiva, A. J.** XML-Based Video Game Description Language. *IEEE Access*, 2020, vol. 8, pp. 4679–4692. DOI: 10.1109/ACCESS.2019.2962969.

13. **Derakhshandi M., Kolahdouz-Rahimi S., Troya J., Lano K.** A model-driven framework for developing android-based classic multiplayer 2D board games. *Automated Software Engineering*, 2021, 28 (2). DOI: 10.1007/s10515-021-00282-1.

14. **Park, H. C., Baek, N.** Design of SelfEngine: A Lightweight Game Engine. *Lecture Notes in Electrical Engineering*, 2020, vol. 621, pp. 223–227. DOI: 10.1007/978-981-15-1465-4_23.

15. **West M.** Evolve your hierarchy [Online] *Cowboy Programming*. Available: http://cowboyprogramming.com/2007/01/05/evolve-your-heirachy/ (accessed: 22.08.2020).

16. **Hocking J.** *Unity in action: Multiplatform game development in C#*. Manning, 2018. 400 p.

17. **Lisovsky, K.Y.** XML Applications Development in Scheme. *Programming and Computer Software 28*, 197–206 (2002). https://doi.org/10.1023/A:1016319000374

Сведения об авторах / Information about authors
Вадим Андреевич Мельников / Vadim A. Melnikov
аспирант / postgraduate student
Сыктывкарский государственный университет им. Питирима Сорокина / Pitirim Sorokin Syktyvkar State University
167001, Россия, г. Сыктывкар, Октябрьский пр., 55 / 167001, Russia, Syktyvkar, Oktyabrsky Ave., 55

Андрей Васильевич Ермоленко / Andrey V. Yermolenko
к.ф.-м.н., доцент, заведующий кафедрой прикладной математики и компьютерных наук / Ph.D. in Physics and Mathematics, Associate Professor, Head of Department of Applied Mathematics and Computer Science
Сыктывкарский государственный университет им. Питирима Сорокина / Pitirim Sorokin Syktyvkar State University
167001, Россия, г. Сыктывкар, Октябрьский пр., 55 / 167001, Russia, Syktyvkar, Oktyabrsky Ave., 55

Статья поступила в редакцию / The article was submitted 12.03.2022
Одобрено после рецензирования / Approved after reviewing 24.03.2022
Принято к публикации / Accepted for publication 25.03.2022