The Journal of Systems & Software 179 (2021) 110985

Contents lists available at ScienceDirect

The Journal of Systems & Software

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

A holistic approach for cross-platform software development☆

J.Z. Blanco [a,b,*], D. Lucrédio [a]

[a] Computing Department, Federal University of São Carlos, Rod. Washington Luís, Km 235, P.O. Box 676 - 13565-905, São Carlos - SP, Brazil
[b] Federal Institute of São Paulo, Campus Piracicaba, Rua Diácono Jair de Oliveira, 1005, 13414-155, Piracicaba - SP, Brazil

ARTICLE INFO

Article history:
Received 3 July 2020
Received in revised form 5 February 2021
Accepted 12 April 2021
Available online 1 May 2021

Keywords:
Cross-platform development
General-purpose language
Model-driven development
User studies

ABSTRACT

Cross-platform development solutions can help to make software available on different devices and platforms. But these are normally restricted to preconfigured platforms and consider that each individual solution is equal or similar to each other. As a result, developers have to resort to native development and build individual solutions, one for each device/platform, that cooperate to deliver the desired global functionality. This article presents an approach that takes advantage of existing solutions and have support for extending and including new platforms, and distributing functionality across devices. The approach is based on a general-purpose language that raises the abstraction level in order to keep the software free from platform details. Automatic transformations produce executable code that can be properly divided and deployed separately into different platforms. The proposed approach was evaluated in four ways. In the first evaluation, an existing cross-platform system was recreated using the approach. The second and third evaluations was conducted with expert and novice developers, who tested the approach in practice. The fourth evaluation introduced support for cross-platform testing. Results have brought evidence supporting the following main contributions: use of a single environment, the ability to reuse similar concepts between platforms and the potential to reduce costs.

1. Introduction

The great number and variety of computing devices, such as smartphones, tablets and smartwatches, creates an ever-growing demand for cross-platform software, that is, a single software system that runs on multiple devices. In this scenario, developers are challenged to handle device-specific characteristics such as types and versions of operating systems, data storage capacity, available resources (GPS, camera, etc.), screen size, and more (Wasserman, 2010). Dealing with these differences at a low abstraction level have an impact on productivity, maintenance and quality, ultimately resulting in cost increases and loss of focus on the business domain.

There are two additional complications to this scenario. First, there is always the risk of having to support new platforms in the near future. The industry has been investing in the use of computing in various objects (Hoffman and Novak, 2018) and this reality should be considered for software engineering. To keep software competitive, in many cases it is important to run on as many new platforms as possible, as soon as possible.

Second, in some cases, the software must have its functionalities distributed between versions for different platforms (Martinez and Lecomte, 2017). Not always the functions of one version for a specific platform should be maintained in other versions. For example, a mobile smartphone is more suitable for taking geotagged photographs, while a computer is preferred for typing or viewing large pictures or maps. Functions distributed across system versions can better explore the capabilities of each device.

The problem is not recent. In the recent past, when smartphones started to appear, Java emerged as a promising solution, in the form of the Java 2 Micro Edition. In theory, a software could be written in a single way (using J2ME) and run everywhere, as long as there was a supporting virtual machine. But even in the realm of mobile phones, which is a very well defined context, different implementations were provided by different vendors, causing software to crash unless extensively tested on dozens of devices (Umuhoza et al., 2015). Also, each manufacturer started to add specific features, such as dedicated sensors or specific controls. Supporting all of these variations in a single framework was very problematic, and developers had to deal with these differences in the code, by adding preprocessing directives or maintaining completely different versions of the software at the same time (Chen et al., 2019). Eventually, when devices became

☆ Editor: Heiko Koziolek.

* Corresponding author at: Computing Department, Federal University of São Carlos, Rod. Washington Luís, Km 235, P.O. Box 676 - 13565-905, São Carlos - SP, Brazil.
E-mail addresses: julianoblanco@ifsp.edu.br (J.Z. Blanco), daniel.lucredio@ufscar.br (D. Lucrédio).

https://doi.org/10.1016/j.jss.2021.110985
0164-1212/

powerful enough, J2ME and the promise of Java as a universal mobile language approach were replaced by Android, a complete operating system (Heitkötter et al., 2013).

Current web technologies such as HTML5 and Progressive Web Apps[1] can help to make software available in any device with a web browser. It is possible to develop responsive interfaces, suitable for different screen sizes, with offline capabilities and background processing. These can make the experience become closer to a native application, but there are limitations. Accessing hardware features such as Bluetooth, Near-Field Communication, GPS and Camera is not ideal (Litayem et al., 2015). There is also a limiting factor regarding local storage in the browser.[2] Finally, there are devices such as smartwatches, that have very limited hardware and do not feature a fully functioning browser. In these cases, native development is the only option.

There are solutions that help to reduce some of these challenges. Hybrid development (Heitkötter et al., 2013) and generative approaches (Czarnecki et al., 2002) can help to unify the different software versions, but they normally have poor support for a variety of current and new platforms. Responsive web development (Marcotte, 2010) have good potential to support different screen types but have limitations regarding native hardware (Litayem et al., 2015). Service-oriented architecture and cloud computing (Moreno-Vozmediano et al., 2017) may help by moving parts of the system out of the devices and facilitate maintenance of each individual service, but device and platform-specific functions still have to be developed to consume these services. None of these solutions solve the essence of the problem, which is the separate development for what should be treated more like a single system. The developers still have to work with multiple individual systems that work together and deal with the technical details inherent to each platform. Also, when adopting a particular solution, the developer is normally restricted to whichever platforms are currently supported by the tool or approach.

In face of these challenges, this article presents an approach where development and maintenance consider a cross-platform system as a single software entity. Therefore, the approach is considered holistic as it covers multiple characteristics and seeks to uniformly solve the challenges associated with cross-platform development. The developer creates the software using a single set of models, using a Generic Purpose Language (GPL) developed to support the approach. The GPL is a programming language with some high level constructs that allow different domain concepts and functions to be specified completely and independently from the platform(s) on which they will run. Through generators, code is automatically obtained for different platforms.

The approach is also holistic in the sense that it is not restricted to a predefined set of supported platforms. Adding new platforms is embedded in the approach through the same GPL, with which platform models can be specified and code be generated. Therefore, the approach can cover a wide range of devices, both current and future ones. Furthermore, the approach allows existing platforms to be extended or modified, to better suit the needs of a particular system.

Finally, the approach can be used to choose in which platform each part or function will be deployed. This facilitates the job of distributing functionality across devices, for example deploying GPS and camera-dependent functions into smartphones only, and larger reports and data sheets functionality into a web application. This is a major difference from many existing solutions, which normally create different versions of the same software for different platforms.

To evaluate the approach, four studies were conducted. A first study was conducted by the researchers, and consisted in a proof-of-concept where an existing commercial cross-platform system was recreated in the platform. A second study involved five experts, who used the approach to create software using the approach. To demonstrate the holistic nature of the approach in supporting different and new platforms, the studies involved deploying the software in the web, Android and iOS, with different storage technologies (HashMap and SQLite), different programming languages (Java, C# and Swift), and even included a new device created solely for this research: an augmented reality (AR) device based on RaspBerry Pi,[3] with simple input/output capabilities. The third study was similar to the second, but involved four developers with low expertise in mobile development. And the fourth study explored cross-platform unit testing.

The results show that the approach can be successfully used to treat cross-platform development as a single software entity. It also highlighted some contributions, the main being the possibility to use a single environment to create a platform-independent representation of the software. The results also demonstrated the ability to reuse similar concepts between platforms. Ultimately, the approach has the potential to reduce costs. The evaluations also pointed out some limitations, the most important being the need for an initial effort to properly create and prepare the platform details. This is important, as most developers have this support already available in current tools. Also, in some cases it was necessary to use the platforms' native IDEs, thus breaking the purpose of treating the software as a single entity in a single environment. Another limitation is the need for additional tests, to make sure the generated code is adequate and can be trusted. The results also point out some implementation details that could be improved in the future. The collected empirical evidence constitutes important contributions for continuing this research in the future until it is ready to reach production-ready tools and environments.

The remainder of this paper is organized as follows. Section 2 presents related work. Section 3 describes the cross-platform development approach in details. Section 4 contains the evaluation and Section 5 presents some concluding remarks and future work.

2. Related work

HAXE[4] and KOTLIN[5] are two examples of languages that can be used to create platform-independent software by generating code for different native languages, including JavaScript, PHP, Pyhton, C++, Java, among others, offering support for different platforms, including Windows, Linux, MacOS, iOS and Android. The IBM RATIONAL RHAPSODY[6] tool enables the development of embedded systems, real-time systems and commercial software through a UML-based visual modeling language or Systems Modeling Language (SysML). GENEXUS[7] is a business tool that uses the MDD (Model Driven Development) concept (France and Rumpe, 2007) to visually model complex systems and has a separate environment for each platform, it also has a language for specifying business rules as an assistance to visual modeling.

Model-Driven APPLAUSE[8] (Dageförde et al., 2016) is an academic approach that provides a declarative DSL written in XText[9]

1 developers.google.com/web/updates/2015/12/getting-started-pwa.

2 developers.google.com/web/fundamentals/instant-and-offline/web-storage/offline-for-pwa.

3 https://www.raspberrypi.org/.

4 haxe.org.

5 kotlinlang.org.

6 www-03.ibm.com/software/products/en/ratirhapfami.

7 www.genexus.com/.

8 github.com/applause/applause.

9 www.eclipse.org/Xtext.

with templates for XTend code generation. But the focus of modeling involves technical domain details rather than conceptual elements, maintaining a low level of abstraction. Miravet et al. (2014) present another academic approach. The difference between them is the use of XML as a modeling language and the use of the device-independent mobile application generation (DIMAG) to generate native code.

In the work of Inayatullah et al. (2019), an MDA-based framework (Model-Driven Architecture) provides UML modeling to express the domain concepts, generating hybrid applications and RESTful Services in Asp.Net. APPIAN[10] and MENDIX[11] are commercial frameworks that support mobile application development and data interoperability between applications. The difference between them is that the latter also considers the web platform using PhoneGap[12] as the basis. Interfaces are modeled visually and business rules and behavior are defined at a high abstraction level through custom tool editors, the result is executable code generated. Following the business line, WEBRATIO[13] (Acerbis et al., 2015) is a model-driven development platform based on Interaction Flow Modeling Language[14] and with a high abstraction level. WEBRATIO uses the Cordova[15] API as a base and considers the same platforms as MENDIX.

Some work uses MDD to address the development of a single application version. Examples are JUSE4ANDROID (da Silva and e Abreu, 2014), MIMIC (Elouali et al., 2014) and the research by Cimino and Marcelloni (2012), which focuses exclusively on graphical user interface for the Android platform. Behrens (Behrens, 2010) presents an environment for textual modeling through a DSL for iOS. Min et al. (2011) and Benouda et al. (2017) use UML to model applications for Windows Phone and generate native code in C#.

Many approaches try to focus only on Android and iOS, which are currently dominating the market. This is the case of academic approaches such as MD2 (Majchrzak et al., 2015), Chen et al. (2019)'s, Sabraoui et al. (2019)'s, AXIOM (Jones and Jia, 2014), Perchat, Desertot and Lecomte's (Perchat et al., 2013), MOPPET (Usman et al., 2017), Taentzer and Vauper's (Taentzer and Vaupel, 2016), (Dageförde et al., 2016)'s, (Vaupel et al., 2018)'s, Rieger et al. (2020), and industry initiatives such as REACT NATIVE,[16] NATIVESCRIPT,[17] Flutter.[18] and XAMARIN[19]

The REACT NATIVE framework, developed by Facebook, allows writing code through web technologies (XML, CSS and JavaScript). Its contribution is the generation of readable native code for both platforms (iOS and Android), where it is possible to add new functionality directly in the generated source code. NATIVESCRIPT enables the development of native applications using a single language. In this case, the developer may choose between JavaScript, Angular[20] or TypeScript[21] and interface customizations through CSS. The difference between REACT NATIVE and NATIVESCRIPT is that in REACT NATIVE the code is transformed into native language, with the possibility of extension by the developer. And in the second, the code is delivered to devices in JavaScript and interpreted natively through the Virtual Machine (VM) of each platform.

Flutter uses the Dart language to model and generate executable code and is considered an interpreted approach. A compiler converts the code into native language that runs on the device, along with an interface rendering engine. In this way, Flutter does not use native elements for rendering the interface, impairing its performance a little. XAMARIN uses the C# language, and can obtain between 75% and 100% of code reuse between platforms. The applications generated by the tool are native, thanks to the access to the APIs of each platform. However, despite the increase in productivity, the tool does not increase the level of abstraction, keeping the developer involved with technical details.

Chen et al. (2019) propose a framework to convert user interface (UI) source code from Android to iOS and from iOS to Android. It uses a classification algorithm to identify the components in one platform and then generates corresponding elements in the other platform. Sabraoui et al. (2019) use MDD to develop the GUI platform independently. A DSL is used to model the interface, programmed using a grammar written in XText. This DSL can be viewed graphically, as UML models, and through code generators the UI code for the platforms is automatically created.

AXIOM (Jones and Jia, 2014) uses UML visual models combined with Abstract Model Tree to represent domain interactions and generic concepts through a Platform Independent Model. Perchat et al. (2013) propose a universal language in which, through a compiler, it is possible to generate native code, but the proposed language has a low abstraction level. MOPPET (Usman et al., 2017) is a tool that uses a mixed approach, involving the concepts of Software Product Lines and MDD, defined by the authors as a product-line model-driven engineering approach for native code generation. Case studies have shown reduced effort and increased productivity.

Taentzer and Vaupel (2016) propose the textual modeling of native applications separated into three submodels: graphical modeling (GUI), data modeling and application behavior (business rules). The main contribution of this work is the generation of native code taking into account information about the context and offline operation.

Dageförde et al. (2016) discuss ways to add the concept of Software Product Lines (SPL) into the MD2 framework. The main contribution of the work is the modularization of the MD2 framework, according to the concepts of SPL, ensuring greater versatility and reuse of modeling artifacts. In the work of Vaupel et al. (2018) an approach that uses visual models is presented and specific business rules can be expressed in a textual language, for the generation of native codes (Andoid and iOS). In Rieger et al.'s work (Rieger et al., 2020), the elements for accessibility on mobile devices are included in the framework MD2, aiming to reduce costs in its implementation.

In summary, both academia and industry are successful in delivering a single way to develop software for multiple platforms, allowing the developer to reuse code across platforms and domains more efficiently, helping to reduce development and maintenance costs. But they have strong ties with the supported platforms, not giving the developer the choice to include new platforms or extend existing support without considerable effort. They also normally consider that all platforms will receive the same functions, not allowing different devices to receive different functions. We formalize all these desired contributions as follows:

C1. High abstraction level modeling in a single environment. The software engineer should be able to specify system concepts at an abstraction level that sits above platform-specific details.

10 www.appian.com/.
11 www.mendix.com.
12 phonegap.com/.
13 www.webratio.com.
14 www.ifml.org/.
15 cordova.apache.org/.
16 facebook.github.io/react-native.
17 www.nativescript.org.
18 flutter.dev.
19 www.xamarin.com.
20 angular.io.
21 www.typescriptlang.org.

Table 1
Brief comparison between related work and this one, in terms of the identified contributions.

Related work	C1	C2	C3	C4	C5	C6	C7
HAXE	X	X	-	-	-	X	X
KOTLIN	X	X	-	-	-	X	X
IBM	X	X	-	-	-	X	X
GENEXUS	X	-	-	-	-	X	X
APPLAUSE	X	X	-	-	-	X	X
MIRAVET Miravet et al. (2014)	X	X	-	-	-	X	X
INAYAT. Inayatullah et al. (2019)	X	X	-	-	-	X	X
APPIAN	X	X	-	-	-	X	X
MENDIX	X	X	-	-	-	X	X
WEBRATIO Acerbis et al. (2015)	X	X	-	-	-	X	X
JUSE4da Silva and e Abreu (2014)	X	-	-	-	-	X	X
MIMIC Elouali et al. (2014)	X	-	-	-	-	X	X
CIMINO Cimino and Marcelloni (2012)	X	-	-	-	-	X	X
BEHRENS Behrens (2010)	X	-	-	-	-	X	X
MIN Min et al. (2011)	X	X	-	-	-	X	X
BENOUDA Benouda et al. (2017)	X	-	-	-	-	X	X
MD2Majchrzak et al. (2015)	X	X	-	-	-	X	X
CHEN Chen et al. (2019)	X	X	-	-	-	X	X
SABRAOUI Sabraoui et al. (2019)	X	X	-	-	-	X	X
AXIOM Jones and Jia (2014)	X	-	-	-	-	X	X
PERCHAT Perchat et al. (2013)	X	X	-	-	-	X	X
MOPPET Usman et al. (2017)	X	-	-	-	-	X	X
TAENTZER Taentzer and Vaupel (2016)	X	-	-	-	-	X	X
DAGEFÖRDE Dageförde et al. (2016)	X	X	-	-	-	X	X
VAUPEL Vaupel et al. (2018)	X	X	-	-	-	X	X
RIEGER Rieger et al. (2020)	X	X	-	-	-	X	X
REACT NATIVE	X	X	-	-	-	X	X
NATIVESCRIPT	X	X	-	-	-	X	X
FLUTTER	X	X	-	-	X	X	X
XAMARIN	X	X	-	-	-	X	X
THIS WORK	X	X	X	X	X	X	X

C2. Reuse of similar concepts between platforms through modeling. Many concepts, such as persistent entities and functions, have to be repeatedly created, with small differences, for each platform. The approach must allow these to be reused in all platforms.

C3. Reduced system configuration costs. This includes distributing functionality among platforms and switching the platforms being used in the system.

C4. Inclusion of new platforms with a small impact. The ability to include new platforms into the system, even if they were not initially supported.

C5. Extension of the approach through platform models. Existing platform support should be extensible to better suit the needs of a particular system.

C6. Code reuse across domains. Cross-domain concepts and functions should be specified in such a way that they can be easily reused in any domain.

C7. Development and maintenance cost reduction. The approach's support for abstraction, reuse and code generation should bring the potential to reduce costs during the development life cycle.

Table 1 presents a simple comparison between existing work and our proposed approach, in terms of these seven contributions. The table is just a summary of the essence of each solution. There may be specific ways to achieve these contributions that were not accounted for because this is not explicitly mentioned in the research paper or documentation. All solutions deliver higher abstraction level development (**C1**), cross-domain reuse (**C6**) and cost reduction (**C7**). But none have been designed to allow functionality distribution (**C3**) and the inclusion of new platforms (**C4**). Platform extension (**C5**) is a recent addition to Flutter in the form of "platform channels"[22], which are somehow similar to this approach's concept of global functions, as discussed later.

3. A holistic approach for cross-platform software development

In order to overcome the limitations of existing work, this research proposes a holistic view of cross-platform software development.[23] For that, it borrows the main idea from Model Driven Development (MDD) (France and Rumpe, 2007), which is to combine generative programming, software transformation techniques and domain-specific languages (DSL). But just using MDD for cross-platform development has already been proposed and successfully tested by others, as discussed in the previous section. The main difference introduced in this research is how platform-specific details are taken into consideration, as shown in Fig. 1.

The left side of Fig. 1 shows a typical MDD-based cross-platform solution. The software engineer works on a higher abstraction level, to produce a platform-independent model. She is shielded from the details of the underlying platforms by means of platform-specific generators and transformations. These artifacts are provided to the developer as black-box components, to be used as they are or with some parameterization. As a result, the developer can focus on business domain concepts, as intended by MDD. But there is little flexibility to modify platform details or include new platforms. The only solution would be to modify the code generators or transformations, but since these were designed to be used as black-box components, this is a difficult task. Also, code generators produce different versions of the same software. This means that all functions for the entire system are generated for all platforms, equally.

In contrast, this approach (right side of Fig. 1) takes a more holistic viewpoint. Business domain concepts and the platform details are meant to be under the control of the developer. The only black-box components are the language-specific code generators, which will only need to change if the target language changes, what is not likely to occur very often. This allows platforms to be modified or added by the developer. The approach also provides the ability to choose where each part of the software will be deployed. For example, an administrative area of a system may be targeted at the web only. The main front-end may be targeted at the web and mobile platforms, and a part of a system dealing with sensors might be targeted at wearable and arduino platforms. All these specifications are platform-independent, and the entire solution is treated as a single software entity.

This is made possible with a Generic Purpose Language (GPL) designed for this goal. This GPL is a textual programming language with some high-level constructs for business domain concepts, detailed behavior, platform details and functionality distribution. So, if a different way of generating code for a specific platform is needed, or if a new platform needs to be supported, the developer can use the GPL instead of dealing with code generators or modifying generated code. Fig. 2 shows the three models that need to be specified by the software engineer when creating a cross-platform system using the GPL: the system model (left), the platform model (right) and the deployment model (middle).

[22] flutter.dev/docs/development/platform-integration/platform-channels.
[23] https://doi.org/10.5281/zenodo.4728219.

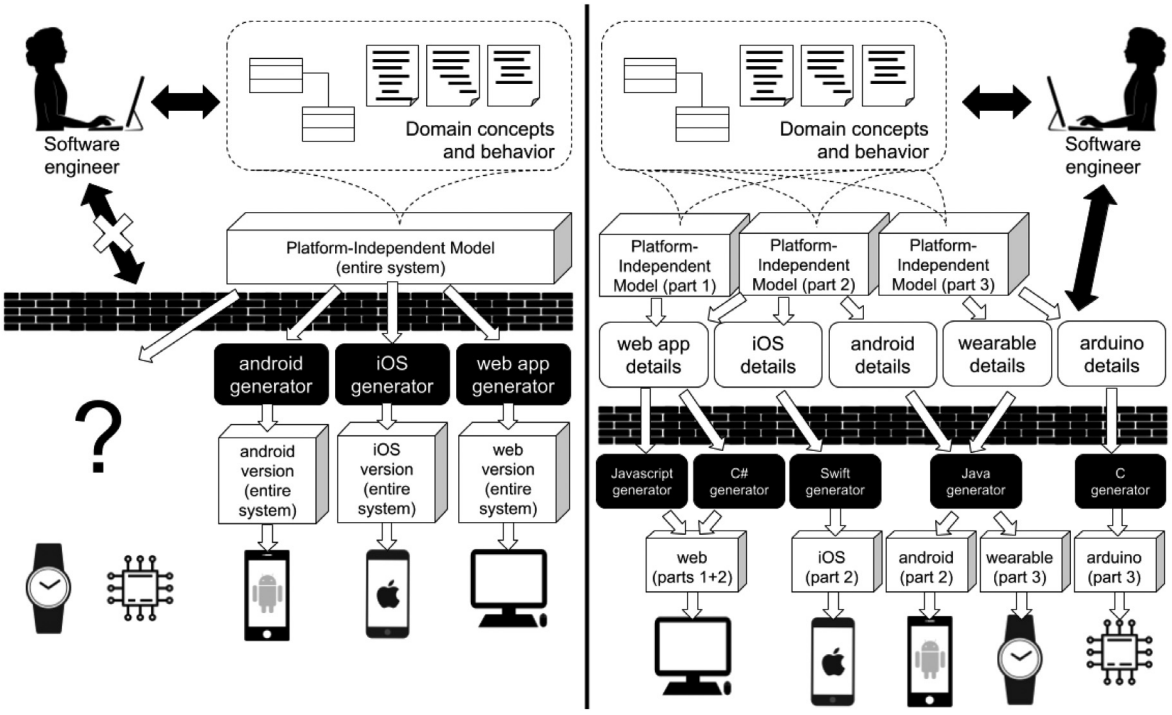

Fig. 1. An overview of a typical MDD cross-platform solution (left) compared with the approach proposed in this research (right).

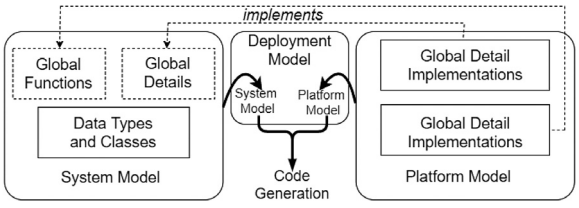

Fig. 2. Cross-platform approach elements.

3.1. The system model

The system model is composed of three main parts: Data Types and Classes, Global Functions and Global Details.

The first part of a system model are **Data Types and Classes**. These are regular object-oriented constructs to specify a system's structure and behavior. Listing 1 shows an example of this part of the system model, where three data types (lines 1–3) and one class (lines 4–12) are defined. The class has two attributes (line 5) and one operation (lines 6–12). Currently, the GPL has support for most object-oriented programming language features, however it is still a prototype and may have some missing implementation details.

Listing 1: Data type and class declarations using the GPL.

```
1  datatype int
2  datatype string
3  datatype double
4  class Coordinates {
5     double x, y;
6     operation dist(o: Coordinates): double {
7        dx: float
8        dy: float
9        dx := o.x - x
10       dy := o.y - y
11       return sqrt(dx^2 + dy^2)
12 } }
```

The second part of a system model are **Global Functions**. These are the most important elements of the approach, as they establish the relationship between platform-independent and platform-specific models. Consider for example a function for obtaining the device location. Each platform (android, iOS, web, etc.) will have a different implementation, possibly with a different function signature. But, in essence, they could be represented as a function, called "getLocation()" that returns a pair of coordinates (latitude and longitude). In the approach, this is exactly what a global function is: an abstraction for a function that is needed by the system, has the same (or a very similar) signature, but requires platform-specific implementations.

Listing 2 shows an example of global function declarations. It is possible to see that the approach supports cross-domain global functions, such as `getGPSPosition` (line 1), to obtain a device's position, for example. This type of function can be used in different applications, from different domains. The approach also supports domain-specific global functions, such as `InsertProductIntoCart`, `SelectCustomer` and `SelectProduct` (lines 2–4). These are specific to an e-commerce domain.

Listing 2: Example of Global Function Declarations.

```
1  global getGPSPosition(): Coordinates
2  global SelectCustomer(id: string): Customer
3  global SelectProduct(id: string): Product
4  global InsertProductIntoCart(customer: Customer,
       product: Product, quantity: int): string
```

A global function is supposed to be supported by different platforms. This is possible as long as the platforms are able to implement the declared function prototype. For example, most smartphones are capable of obtaining their positions from GPS satellites, therefore they will probably be able to provide a compatible implementation for function `getGPSPosition`. But a desktop or portable computer does not have a GPS receiver, therefore it will not be able to implement this function.[24] As

[24] Computers have location systems, but they normally depend on antennas and might not function in a farm or in the ocean, for example.

discussed before, the approach does not assume that all platforms will run identical versions of the entire system, therefore not all global functions will be implemented by all platforms. Instead, the approach assumes that the system requires that these functions will be implemented by at least one platform, otherwise the system will not be able to execute properly.

The third part of a system model are **Global Details**. These represent platform-specific code that only makes sense in a particular platform and that cannot be generalized as a global function. In other words, any piece of code that is not the implementation of a global function, such as additional variables, methods and annotations, that must somehow appear in the generated code, must be specified as global details. For example, global function `SelectCustomer` from Listing 2, when deployed in the web platform, requires a database connection to be available. The implementation for this function could contain code that creates a new connection every time the function is called, but a more efficient approach would be to create the connection once, store it as a class attribute, and reuse it across calls. A class attribute would fall outside the implementation of a global function, and this is where global details are useful. They can be used to include class attributes, additional methods (other than the global function itself), annotations, among other code.

Global details consist of a declaration, implementation and use. The declaration is very simple, as shows Listing 3. The implementation of this example, which will create a connection attribute and helper methods, will be described later (Listing 7).

Listing 3: Example of Global Details Declarations.

```
1  globalDetails DBConnection
```

Global functions and details can be used by any class. An example is shown in Listing 4. The use of a global details in a class is declared with the keyword `usesGlobalDetails` (line 2). In this example, these details represent database connection initialization code. The use of a global function in a class is declared with the keyword `usesGlobal` (lines 3–5). To improve readability, the complete global function declaration has to be repeated into the class. Then, these functions can be invoked normally (lines 11–13) as if they were local functions, except for the keyword `global`.

Listing 4: A class using global functions and details.

```
1  class ShoppingCartDAO {
2      usesGlobalDetails DBConnection
3      usesGlobal SelectCustomer(id: string):
           Customer
4      userGlobal SelectProduct(id: string): Product
5      usesGlobal InsertProductIntoCart(customer:
           Customer, product: Product, quantity: int
           ): string
6      operation addProduct(custId: string, prodId:
           string, quantity: int): list<string> {
7          objCust: Customer
8          objProd: Product
9          newProd: string
10         accessData: list<string>
11         objCust := global SelectCustomer(custId)
12         objProd := global SelectProduct(prodId)
13         newProd := global InsertProductIntoCart(
               objCust, objProd, quantity)
14 } }
```

Each global function will have a different implementation on each platform. In the example of Listing 4, the global invocations in lines 11–13 might refer to a SQL database if the class is deployed into a web platform, or to a SQLite database if the class is deployed into a mobile platform. But here in the system model this is intentionally left undefined. It shouldn't matter which platform will run this code, it only needs a compatible implementation for these functions.

Global functions and global details are very similar to the concept of platform channels present in Flutter. There are two main differences: first, in Flutter only the predefined platforms are supported, while in this approach any platform can implement any global function; second, in Flutter, channels are linked to the implementation during runtime, while here they are included in the generated code, thus being linked during compilation time. Flutter's runtime-based approach makes it easier to reuse channels as black-box components, in different apps. This approach's generation-based process, together with global details, makes it easier to extend or customize platform-specific code.

3.2. The platform model

The platform model consists of implementations for the global functions and global details, and is defined for a particular language. The global function prototypes are declared using the GPL syntax, but the actual implementation of these functions may be written using either the GPL or the platform's native programming language. Using the platform language is preferred, as it allows more direct access to all of the platform's features. Listing 5 shows three example of platforms: Web using C# (lines 1–5), iOS using Swift (lines 6–10) and Augmented Reality using C# (lines 11–15), an entirely new platform developed for this project.

Listing 5: Implementation of global functions.

```
1  platform Web: CSharp {
2    implementsGlobal InsertProductIntoCart(c:
         Customer, p: Product, quantity: int):
         string {
3      //C# code that implements the function on the
           web
4      //platform using SQL for persistence
5  }  }
6  platform iOS: Swift{
7    implementsGlobal InsertProductIntoCart(c:
         Customer, p: Product, quantity: int):
         string {
8      // Swift code that implements the function on
           iOS
9      // using SQLite for device persistence
10 }  }
11 platform AugmentedReality: CSharp {
12   implementsGlobal GetDiskImagesAsync(
         picturesFolder: StorageFolder): list<img>
         {
13     //C# code that implements the function on
14     //the platform for Augmented Reality
15 }  }
```

As discussed before, the holistic approach does not assume that all platforms will run identical versions of the entire system. In the example of Listing 5, it makes no sense to provide an implementation for the `InsertProductIntoCart` global function for the Augmented Reality platform, as this platform does not have data persistence support. Therefore, only the Web and iOS platforms will contain an implementation for this global function. The Augmented Reality platform will have an implementation for global function `GetDiskImagesAsync`, for displaying images.

The software engineer has flexibility to provide custom implementations for each platform. In the example of Listing 5, each platform adopts a different persistence mechanism: SQL-based, for Web/C# platform (lines 3–4), and SQLite-based, for the iOS/Swift platform (lines 8–9).

Also to increase the flexibility, the approach has support for Apache Velocity templates,[25] making implementation code more

[25] velocity.apache.org.

flexible and increasing reusability. As an example of this additional feature, Listing 6 shows a domain-independent way to declare, implement and use a global function `SelectObject`, which can query any object from a SQL database. In the declaration part (line 2), there is nothing different from a normal global declaration, except for the generic type <E>, which must be specified later.

In the implementation (lines 4–21), triple apostrophes (''' in lines 5 and 21) are used to delimitate the template contents. Apache Velocity tags (starting with # and $) are used to query the generic type <E> in search for its features. It is possible, for example, to use E's name ($E.name in line 12) to generate an SQL statement. When using this global function, it is necessary to specify a concrete class to be used for code generation. In this example, class `Order` (line 23) will be the object being queried. The approach takes care of linking these definitions and generating correct code for the platform. The result is that this global function may be reused in different domains.

Listing 6: Declaring, Implementing and Using a domain-independent global function using Apache Velocity.

```
1  //Declaration
2  global <E> SelectObject(ord: string): list<E>
3  //Implementation
4  implementsGlobal <E> SelectObject(ord:string):list
       <E>{
5  '''
6     List<$E.name> listObj = new List<$E.name>();
7     $E.name obj = new $E.name();
8     Connection objCon = new Connection();
9     MySqlConnection Conn = new MySqlConnection();
10    Conn = objCon.OpenConnection();
11    MySqlCommand command = Conn.CreateCommand();
12    command.CommandText = "select #foreach($f in $E
          .attributes)${f.name}#if$(foreach.hasNext)
           , #end#end from $E.name " + ord;$
13    for(Reader.Read()){
14        #set($count = 0)$
15       #foreach($f in $E.attributes)
16          obj.set${f.name.substring(0,1).
                toUpperCase()}${f.name.substring(1)
                }(Reader.get${f.type.name.substring
                (0,1).toUpperCase()}${f.type.name.
                substring(1)}($count))$;
17          #set($count = $count + 1)
18       #end
19       listObj.add(obj)
20    } return listObj;
21 ''' }
22 // Use
23 usesGlobal <Order> SelectObject(ord: string): list
       <Order>
```

In addition to global functions, the platform model also contains implementations of global details. Listing 7 provides an example of an implementation of global details. As discussed before, these are used to create platform-specific code (fields and methods) that cannot be generalized as global functions. In this case, connecting to MySQL web platform in C# requires a class attribute for the connection (line 4), a method for opening the connection (lines 5–9) and a method for closing the connection (lines 10–12). These will be used internally by global function implementations.

Listing 7: implementation of global details for a MySQL database connection in a web platform.

```
1  platform Web : CSharp {
2      implementsGlobalDetails DBConnection{
3      '''
4          private MySqlConnection Conn;
5          public MySqlConnection OpenConnection(){
6              Conn = new MySqlConnection("server
                   =127.0.0.1;database=ecommerce;uid
                   =18feb24ac5h;pwd=pass");
7              Conn.Open();
8              return Conn;
9          }
10         public void CloseConnection(){
11             Conn.Close();
12         }
13     '''
14 }
```

If a platform has no need for specific details, it may leave the implementation for the global details empty. As a result, no additional code will be generated for that platform.

3.3. The deployment model

The deployment model simply defines which classes will execute on which platforms, serving as a guide for code generation. The snippet shown in Listing 8 presents an example of deploying different classes on different platforms. The only restriction is that a platform chosen for deployment of a particular class must have an implementation for all global functions and details used by that class. In this example, the `User` and `UserDAO` classes are being deployed on all platforms. The `ShoppingCartDAO` class is being deployed on Android (line 8) and iOS (line 9), but not on the web.

Listing 8: Deployment Configuration (deploy).

```
1  deploy Ecommerce {
2      User: Android
3      User: iOS
4      User: Web
5      UserDAO: Android
6      UserDAO: iOS
7      UserDAO: Web
8      ShoppingCartDAO: Android
9      ShoppingCartDAO: iOS
10 }
```

3.4. Code generation

As discussed before, the only black-box components are the language-specific code generators. Currently, there are generators for the Java, C#, and Swift languages, so it is possible to define platforms based on these languages. New generators can be added to the approach by implementing a mapping between the GPL constructs and target language constructs, using Eclipse Xtext[26] and Apache Velocity[27].

Fig. 3 shows an example of deploying classes `User` and `UserDAO` into the web/C# platform. It is possible to see how the system and platform models from the examples shown in this section are integrated into the generated code, which will contain all that is necessary for proper execution.

4. Evaluation

Four evaluations were conducted with the goal of finding evidence regarding the seven main contributions of the approach presented in the end of Section 2. The **first** evaluation was a proof-of-concept, focused on verifying if the approach can be used to produce a real cross-platform system. The **second** evaluation was an expert evaluation. It was conducted with five experts and focused on gathering specialized opinions regarding the expected contributions. The **third** evaluation was similar to

[26] www.eclipse.org/Xtext/.
[27] velocity.apache.org.

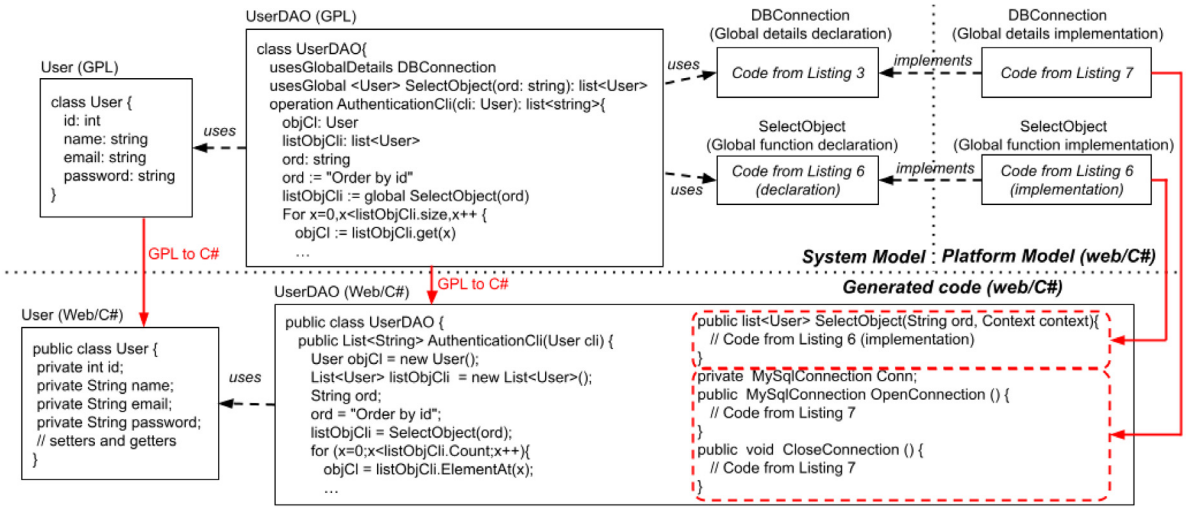

Fig. 3. Code generation result for classes User and UserDAO in the web/C# platform.

the second, but with novice developers. The idea was to test if the approach can shield these developers from the technical details, allowing them to develop software for these platforms even without the necessary knowledge. Finally, the **fourth** evaluation involved using the approach to create test cases for different platforms, following observations made by the experts.

4.1. Proof-of-concept

This evaluation was conducted by the researchers, and had three stages. The **first** stage occurred before the approach was developed, and consisted in the development of a complete cross-platform system for the e-commerce domain. Since the approach did not exist then, the development followed an ad-hoc process. Three platforms were supported: web, android and iOS. The system's architecture follows the Model View Controller (MVC) (Bucanek, 2009) style,. The system has basic CRUD (Create, Retrieve, Update and Delete) functions, an area for customers to browse and order products using a virtual shopping cart, and an administrative area for sellers to process the orders. The customer area also includes functions to detect the customer location. Mobile platforms (Android and iOS) only have the customer area, while web platforms have both the customer and administrative areas.

The **second** stage occurred after the approach was developed, and consisted in migrating the existing system into the approach, i.e. creating the system, platform and deployment models. The main challenge in this stage was to decide which parts of the system would become system models (data types and classes) and which would become platform models (global functions and global details). As discussed before, the main idea is to try to generalize common functions with similar signatures as global functions.

Device location support was obviously part of the platform models, so it was defined as global function and details. For the other parts, we identified the similarities between the platforms in the original code. We observed that around 79% of the developed system consists in concepts and functions that are related either to the MVC architectural style or to CRUD functions. Therefore, we defined global functions and details for these concepts, and used Velocity templates to implement them. As a result, two platforms for each device type were defined: one for the "Model" layer and one for the "Controller" layer. For this proof-of-concept, we did not migrate the "View" layer into the approach, as it involves visual design and layout definition that is better performed directly in the platform's native IDE.

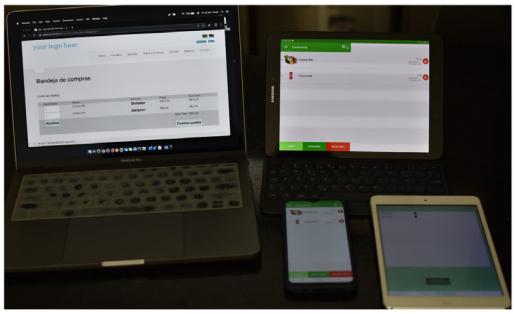

Fig. 4. Shopping cart screen of the e-commerce application shown on four real devices, in three different platforms.

Fig. 5. Augmented reality device highlighting real-world products and projecting their names above as they are selected in the e-commerce application.

In the **third** stage, a fourth platform was included, to test the approach's flexibility to include new platforms. To really put this flexibility to the test, a new platform was created solely for this research, so that it would be demonstrated that the approach does support unforeseen devices. The platform consisted of an augmented reality (AR) device that can project images on real-world objects, allowing customers to see products on a display being highlighted by the device.

Figs. 4 and 5 show five different devices running the application. In Fig. 4 four devices are showing the shopping cart screen: a desktop computer for the web platform, two tablets (Android and iOS), and one smartphone (Android). Fig. 5 shows the augmented reality device projecting images over some products on display, to highlight them as they are selected by the customer.

Table 2
LOC analysis for the proof-of-concept.

		Without approach	With approach GPL	Native
Platform-specific code	Android	1204	16	207
	iOS	1503	16	157
	Web	1012	10	97
	AR	0	55	89
Platform-independent code		0	238	0
Total		3719	335	550

4.1.1. Results

With the proof-of-concept developed and running, the seven contributions of the approach were evaluated.

Table 2 shows a comparison in terms of LOC (lines of code), without and with the approach, excluding the "view" layer of the MVC architecture as it was not implemented in the proof-of-concept. The three zeros in the table are explained as follows: (1) there is no platform-specific code for the augmented reality (AR) device without the approach, as it was only included in the second stage; (2) there is no platform-independent code without the approach, as it is written entirely in the platform's language and cannot be reused in the other platforms (Android, iOS and Web platforms use different languages in the proof-of-concept); and (3) there is also no platform-independent code written in a native language with the approach.

When comparing the total size of the two versions of the system (last line of Table 2), there is a large reduction in terms of LOC. This is expected, as the generic CRUD functions were made generic and were reused for all persistent classes. This observation indicates that contribution **C7** (Development and maintenance cost reduction) is being delivered by the approach. The LOC analysis also indicates that the reuse of similar concepts between platforms through modeling is made possible with the approach (contribution **C2**). Each class was deployed in two or three platforms. This also contributed to the reduction in LOC.

Finally, the LOC analysis indicates that the approach leads to a better focus on conceptual work. While all 3719 lines of code of the original system mix generic domain concepts and platform details, with the approach there are 238 lines of code that correspond exclusively to high-level concepts. The remaining lines of code, created using the GPL and native code, focus on platform details alone. This separation helps to raise the abstraction level in which the software engineer can work (contribution **C1**).

The proof-of-concept have also shown that distributing functionality across the platforms and devices is possible and requires little effort (contribution **C3**). It was very simple to replicate the original system deployment with the approach, with the added benefit of avoiding code duplication.

The third stage of the proof-of-concept demonstrated the ability to include new platforms (contribution **C4**). A newly created device for augmented reality was included in the system without requiring an entirely new development. Existing classes were targeted at the AR platform during deployment and new global functions were successfully added.

In summary, the development of the proof-of-concept led to indications that all contributions, except **C5** (extension of the approach through platform models) and **C6** (code reuse across domains), are being achieved with the approach.

4.2. Expert evaluation

A second evaluation consisted in an experiment involving experts with large experience in web and mobile technologies (minimum 10 years) and some experience with multiplatform frameworks (desirable but not mandatory). Using convenience sampling, the researchers searched through their contacts in academia and industry in search for volunteers. In the end, five experts that fulfill the requirements were selected. All of them have experience in different programming languages, system architectures, programming languages and databases. Two experts (**E1** and **E2**) work in software companies, and have large software development experience. Two experts have academic and research experience, one being a PhD candidate in computer science (**E3**) and one being a PhD teacher in a computer technology center (**E4**). The fifth expert (**E5**) has a PhD in computer science and has experience in the industry but is currently a teacher and researcher in a technology center.

In this evaluation, the experts had to perform five different software development tasks using the approach. The tasks were defined as follows:

Task 1: implement a shopping cart for a restaurant application for three platforms (Android, iOS and Web).

Task 2: reconfigure how each part of the proof-of-concept system is deployed in the different platforms. The experts were also asked to use the GPS location in some specific platforms.

Task 3: include the support for a new database into the proof-of-concept.

Task 4: create a new project, for a different domain (a library), with functions for user authentication and borrowing books.

Task 5: include a new platform (Android smartwatch) in the e-commerce proof-of-concept, where the product listing functionality would be deployed.

In all tasks, experts could reuse all models and code from the e-commerce proof-of-concept.

The evaluation with experts had three stages. In the first stage, the experts had to learn how to use the approach and how the proof-of-concept system is organized. A repository containing documents and tutorials[28] was made available for the experts. The repository also has the source code for the proof-of-concept system, including all models (system, platform and deployment). The code was fully documented to facilitate understanding. Experts were asked to record the time spent in the first stage.

In the second stage, the experts had to perform the tasks. They used their personal computers, using their own free time. They were also free to contact the researchers to ask questions. The time spent in each task should be recorded.

In the third and final stage, each expert was interviewed. A script with eleven questions (**Q1**-**Q11**) was defined:

Q1. What is your opinion regarding the learning curve of the approach?

Q2. What is your opinion regarding the programming language (GPL) used in the approach? Is the language easy or difficult to use? Is it intuitive? How complex is the language?

Q3. What is your opinion regarding the platform models provided by the approach? Is it possible to include all technical details through global functions? Is it possible to reuse technical details in systems from different domains? Is it possible to extend the approach through platform models?

[28] The complete experiment material is available at: https://doi.org/10.5281/zenodo.4728219.

Q4. What did you think of the global functions and generic parameters, used together with Velocity templates, to create dynamic functions in the platform models? Can they automate code generation? Do they allow the creation of dynamic functions during coding? Do they have the potential to create custom code for different entities?

Q5. What did you think of the deployment model? Comment on: the distribution of functionality, switching platforms for some functions, such as receiving orders in the examples, and managing code generation through this model.

Q6. Do you believe the approach can reduce cross-platform development costs?

Q7. How do you evaluate the maintenance of a cross-platform system developed with the approach? Were you able to change the modeling (GPL) and reuse the generated code? For example, switching a variable from float to double or change the way the shopping cart stores information (in a database or in memory).

Q8. How do you evaluate the benefits provided by the approach in the development of cross-platform systems?

Q9. Please rate your experience with the approach in a scale from 1 to 10, where 1 means a very poor experience and 10 means a very good experience.

Q10. Do you have another idea to reduce costs in cross-platform development?

Q11. Please rate each of the following contributions in a scale from 1 to 10. (all contributions, except C7, are listed for the expert to rate)

Most questions were open-ended. Therefore, in order to analyze the responses, the interviews were recorded and a transcript was written to facilitate analysis.[29] The transcripts were analyzed separately by the two authors of this paper. During this analysis, the researchers annotated those parts that indicate some evidence in favor (**C+**) or against (**C-**) one or more of the contributions. The two sets of annotations were then compared and a common agreement was reached after discussions. Finally, the number of annotations was counted in order to provide a quantitative view of the collected evidence. In this process, when there were more than one annotation of the same type in the same response from the same expert, only one was accounted for. This was necessary because sometimes the experts were repeating themselves during their responses to a question.

4.2.1. Quantitative analysis

All tasks were completed successfully by all experts. Table 3 summarizes the time spent during learning and during each task, for each expert. As it can be seen, the average time spent by experts during the entire study was 14h10. Learning took, in average, 6h24, with the remaining time spent in the five tasks. There was some variability from expert to expert, regarding some tasks. Expert **E3** was the fastest in most tasks, and task **T2** was the longest, followed by **T5**. Despite these differences, we consider that the experts were not significantly different from each other and from what was expected, indicating that the tasks were performed more or less in the same way by the experts.

Question **Q9** asked experts to rate the approach, which received an average rate of 9. Question **Q11** asked experts to analyze each contribution. The average ratings were the following: C1=9.2, C2=10, C3=8.8, C4=9, C5=9.2 and C6=8.4. Overall, all experts provided very positive results for all questions, which might represent some bias and is a possible validity threat, as discussed later. But it also indicates that these contributions are being achieved, in particular **C2**, which received maximum rating from all experts. Some smaller values were provided by some experts in particular cases. These are analyzed in the following sections.

Table 3
Time spent by each expert in learning and in each task.

Exp.	Learn.	T1	T2	T3	T4	T5	Total
E1	6h00	1h00	2h20	0h40	0h45	1h30	**12h15**
E2	4h00	0h40	3h40	0h45	0h30	4h00	**13h35**
E3	7h00	0h25	1h20	0h35	0h25	1h30	**11h15**
E4	8h00	2h30	2h20	2h30	0h30	2h00	**17h50**
E5	7h00	2h00	2h55	1h30	1h30	1h00	**15h55**
Ave.	**6h24**	**1h19**	**2h31**	**1h12**	**0h44**	**2h00**	**14h10**

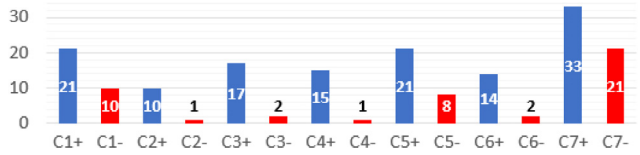

Fig. 6. Number of annotations in favor (**C+**) and against (**C-**) each contribution.

Fig. 6 shows the total amount of annotations for each contribution, after an analysis of the responses. As it can be seen, in general, all contributions have a positive result, i.e. with more annotations in favor than against them.

Although the quantitative results are overall positive, there were limitations and negative observations made by experts. In the following section each contribution is individually analyzed, based on the expert responses.

4.2.2. Qualitative analysis

A total of 21 annotations favor contribution **C1 (High abstraction level modeling in a single environment)**. 33% of the annotations regarding **C1** appear in question **Q2**. Four experts (**E1**, **E2**, **E3** and **E4**) were able to identify the higher abstraction levels and hiding of technical details behind the platform models. Among these 21 annotations, the following excerpts from the transcript exemplify this perception:

E2: "The complexity is low because most details are concentrated in the platform models (...) It is possible to keep the logic in a high level, using the GPL.". **E3**: "The approach increases the abstraction level and considerably reduces the complexity".

There were also 10 noticed limitations for contribution **C1**. Four experts (**E1**, **E2**, **E3** and **E5**) reported the lack of a way to test the algorithms written in the GPL. This is not a problem with the approach, as it is possible to create a platform for generating test code, but the experts failed to see this possibility. Expert **E2** also emphasized, in question **Q2**, that the GPL could be improved with syntax validation and code completion. There are some basic functions in this regard provided by Xtext, but improvements are necessary to make the current implementation more usable.

Another limitation pointed out by the experts (**E2** in **Q6** and **E5** in **Q10**) is a lack of confidence in the generated code. They did not point out any error during the experiment, but they raised the question, which indicates that they might not trust code generation. Expert **E2** also pointed out possible performance problems in the generated code. Although most of these are probably solvable through GPL programming and platform optimizations, the concern reinforces this lack of trust. This has a

[29] The complete experts transcripts are available at: https://doi.org/10.5281/zenodo.4728219.

negative impact on contribution C1, as the developer will have to inspect and test the lower abstraction level models and generated code to be more confident.

There are two sources of possible errors in code generation: in the code generators and in the platform models. The code generators might have some problems, but they should disappear as the approach is used more often and bugs are discovered. The platform models may also introduce errors, and they must be tested to increase confidence. As pointed out by **E5**, the approach does not provide a good support for native programming, and the developer will have to resort to the native IDE. This also goes against C1.

Expert **E1**, in **Q10**, reported difficulty in creating some of the classes for the "controller" layer. Having to customize these classes for each platform, through the concept of global details, required a new way of thinking. According to the expert, this was difficult to grasp at first, but once the concept is fully understood it may lead to long term benefits.

Expert **E4** reported in **Q10** the lack of support for the "View" layer. This was expected, as this layer not included in the platforms provided for the study.

According to expert **E3** in **Q10**, the higher abstraction level would benefit from a visual representation of the classes. Since the GPL is a textual language, developers might find it difficult to understand the relationship between the classes. This is also a limitation related to C1.

Now we analyze contribution **C2 (Reuse of similar concepts between platforms through modeling)**. There were 10 annotations in favor of this contribution, with around 30% appearing in **Q4**. These are indications that the approach can facilitate the reuse of similar concepts between platforms. Some examples extracted from the transcript illustrate the experts' opinions: **E1**: "(The approach) can use the same algorithm to generate code for different platforms." **E2**: "If the GPS example worked for Android and iOS then it works with any other resource."

In question **Q7** the experts pointed out advantages that come from reusing the source code in different platforms during system maintenance, with changes being performed at high level in the GPL (**E1** and **E4**). For example, expert **E4** stated that "All you have to do is save the changes and the native code for every platform is updated." Experts also mentioned the possibility to reuse generic algorithms for different platforms in **Q5** (**E2** and **E3**).

The reuse of similar concepts between platforms, however, was perceived as a medium to long term benefit, as it is necessary to develop the platform models first. For example, **E3**, in **Q2**, stated: "I believe the benefits will only be reached in the long term, when all the platform models and global functions are implemented". According to **E3** and **E5**, having to manually create and test, in their native IDEs, all the global functions for a cross-platform system requires a lot of effort. This initial effort can be reduced as new systems reuse existing models. Importing older models was cited as a possible way to overcome this limitation, as mentioned by some experts (**E5** in **Q7**, **E1**, **E3** and **E4** in **Q6** and **E5** in **Q4**. Expert **E1** even suggested that a collaborative repository could be made available to facilitate reuse.

Contribution **C3 (Reduced system configuration costs)** was unanimously reported in **Q5** as a positive contribution. Experts pointed out some advantages in using a model specifically to configure code generation:

E1: "It is possible to easily see where each class is directed (which platform) and manage functionality distribution". **E3**: "I found it very simple, very interesting, since we can leave everything in a single block and more easily manage the multiplatform system".

However, some limitations were pointed out. As stated by **E3** in **Q2** and **Q8**, and **E5** in **Q5**, the benefits of the deployment model depend on the existence of implementations for the global functions: **E3**: "The approach separates the development among the various abstraction levels involved in the process. However, I believe the gains will be achieved in the long term, when the platform models and global functions are implemented".

Another limitation associated with **C3** was highlighted by **E3** in **Q3**, regarding large systems and library dependencies. In these systems, libraries may become deprecated or suffer changes that require the platforms to be constantly updated. In the studies no such problems appeared, but the fact that an expert demonstrated concern in such scenarios is a potential limitation that must be addressed.

Annotations regarding contribution **C4 (Inclusion of new platforms with a small impact)** appeared predominantly in questions **Q3** and **Q5**. According to the responses to these questions, the approach supports the inclusion of any new function, and any platform that can be developed in one of the supported languages can be included. For example, **E2** says: "I think it is possible to add any technical function in the platform models".

Regarding the impact of such inclusion, experts **E1**, **E4** and **E5** in **Q5** report that it is low, as long as there are some models and implementations from other platforms to be reused. If a new platform has to be completely developed from scratch, there will be considerable effort, as reported by expert **E1** in **Q11**. Knowing how to explore reuse is a key factor, according to this expert.

In order to analyze contribution **C5 (Extension of the approach through platform models)**, we asked experts to extend one of the platforms in Task **T3**, to give them the basis to form an opinion. All experts stated that the approach allows the evolution of the platform models by themselves, without having to wait for new releases from a third-party tool vendor. Another point of interest, made by **E2** and **E3** in **Q3** is the fact that the developer can adjust the generated code. This is an advantage over existing frameworks, as discussed earlier. For example, **E2** says: "Flutter generates code that cannot be changed. Your approach does not have such problem, since it is possible to extend platform models and work with a readable generated code".

This particular comment from **E2** is significant, as it is one of the main benefits expected for the approach. In his comment, the expert cited some problems regarding the differences in configuration and permissions for using the camera and GPS in the iOS and Android platforms. According to **E2**, Flutter does not allow fine adjustments in each platform, treating both platforms in the same way. He also cites specific requirements for publishing applications in the app stores, which may also require adjustments on the generated code for each platform. Since it is not possible to modify the generated code with Flutter, this may cause hard to solve problems during the execution or publication of the application. This approach, on the other hand, allows these changes to be made by the software engineer on the platform models.

The negative comments on this contribution refer to the lack of trust on the generated code. According to **E2** in **Q10**, there must be some way to guarantee that the approach generates code that is correct and with good performance. As discussed in **C1**, the software engineer has control over most of the generated code. Although this brings flexibility, it has a downside, since she is also responsible for testing it and make sure the code is working as expected.

Another limitation mentioned by **E1**, **E2** and **E3** in **Q10**, and **E3** and **E5** in **Q3** is the absence of a test mechanism for new functions in the platform models. This is a negative factor for **C5**, and is similar to the limitation mentioned regarding **C1**. But here the problem is different, since it refers to testing the platforms, and not the system. Ideally, this should be made in the native language, using a native IDE, which goes against the purpose of

using a single environment to create and evolve the software. On the other hand, simpler adjustments and extensions can be made in the models, as reported by **E4** in **Q8**.

The annotations regarding **C6 (Code reuse across domains)** appear in **Q3** (35%) and **Q4** (65%). In **Q3**, some experts reported that the *Apache Velocity* templates help to make the platform models generic enough to be used in other domains. In **Q4** all experts reported that the generated code can be fully customized through global functions, and this allows the platform models to be used with any entity and in any domain. The following excerpts from the transcript illustrate their opinions: **E2**: "I believe it is possible to reuse the code and it is very practical, since it is generic. You can change details very easily, just informing which entity is to be persisted". **E5**: "Global functions make a lot of sense in a multiplatform scenario, but it still makes sense for a single, specific platform".

Expert **C4**, in **Q10**, also pointed out the limitation that the reuse was restricted to the "Model" and "Controller" layers of MVC. This was expected, as in the studies the "View" layer was left out of scope. Although in theory this could be implemented as a new platform and a set of global functions/details, we believe there are many details that need to be addressed, which is why this is left to future work.

The last contribution **C7 (Development and maintenance cost reduction)** was the most cited by experts in their responses, mostly because this is a generic statement that may be influenced by all other contributions. It was even cited as one of the main benefits of the approach by experts **E1** and **E3** in **Q8**.

The annotations regarding **C7** are also generic, and experts associate cost reduction with gains in productivity, management and time. The main reasons for such gains come from: the GPL (**E4** and **E5** in **Q2**); the reuse of platform models (all experts in **Q4**, **E5** in **Q2** and **Q5**, **E4** in **Q5** and **E2** in **Q3**; the overall architecture (all experts in **Q6**); facilitated maintenance (all experts in **Q7**).

The following examples illustrate the perceived benefits in cost reduction: **E5**: "If the global functions are implemented, productivity is greatly increased". **E3** in **Q8**: "It also improves the management of the developers' roles in the process". **E4**: "Time and management gains seems good".

Overall, the opinions strongly support this contribution. Some limitations were cited, however. These appear in all questions, but most are in **Q10**. Most were already discussed previously: the lack of support for the "View" layer, for example, is one of them. The extra effort needed for implementing the platform models was also cited. Experts **E1**, **E3** and **E4**, in **Q6**, explicitly state that the benefits are only possible under the condition that all platform models are implemented. Expert **E5** made comments in this regard in almost all questions. Expert **E3** also points out the need for extra testing for the platform models.

Another limitation is that the approach requires experts in different languages for the proper implementation of all global functions for all platforms (**E3** in **Q4**). However, the same expert indicates that, in the end, this pays off in terms of productivity gains (**Q4**) and better role management (**Q8**).

4.3. Novice evaluation

The third evaluation was similar to the second, but it involved novice developers, not familiar with web and mobile development. We selected four undergraduate students from the last year of their System Analysis and Development course offered at Federal Institute of São Paulo (IFSP) – campus Piracicaba – Brazil. These students had basic programming knowledge and minor experience with IDEs, but they still had not studied mobile and web development. Since the goal of this evaluation was to test if these students could develop software for web and mobile platforms, all the necessary platform models were provided to them prior to the study. As in the previous study, participants were asked to perform some tasks for the e-commerce domain, but here the tasks did not require understanding or modifying the platform models (global functions and global details):

Table 4
Time spent in learning and in each task.

Nov.	Learn.	T1	T2	T3	T4	Total
N1	10h00	2h30	1h00	3h00	0h30	**16h40**
N2	12h00	3h00	1h30	3h00	1h00	**20h30**
N3	7h00	2h00	1h30	3h35	0h40	**14h05**
N4	8h00	4h00	1h30	2h30	0h50	**16h10**
Ave.	**9h15**	**2h50**	**1h20**	**2h55**	**0h45**	**16h48**

Task 1: create an authentication function for a mobile app.

Task 2: create a function to retrieve products stored in SQLite database to present to the end user.

Task 3: add support for updating products in a shopping cart.

Task 4: reconfigure how each part of the application is deployed in the different platforms, and use GPS location in some specific platforms.

All tasks required creating classes for the "Model" and "Controller" layers, with business logic being specified using the GPL. Reuse of global functions and global details was expected. As in the previous study, the "View" layer was already provided and did not require development.

This evaluation also had three stages, as the previous one. A repository with all necessary material (documentation, tutorials and necessary tools) was made available for the participants.[30] The same interview questions used in the previous study was used here, but questions 3 and 4, which focused on understanding and modifying platform models, were not considered as relevant because beginner participants were not expected to be able to answer them properly. All interviews were transcribed and analyzed by the two researchers, as in the previous study.

4.3.1. Quantitative analysis

All tasks were successfully completed by the novice participants. Table 4 presents the time spent by each one. The average time was 16 h 48 min, with an average learning time of 9 h 15 min. These values were higher than those spent by the experts (Table 3), even considering that experts had more tasks to do (modifying platform models). This is expected due to the lack of experience of these participants.

The ratings provided in questions **Q9** (average=9.25) and **Q11** (C1=9.5, C2=10, C3=9.5 and C6=9.5) were positive, indicating that the approach was seen as mostly positive by novice developers, delivering the expected contributions, except for C4 and C5 which were not considered here.

Fig. 7 summarizes the amount of annotations for each contribution made in the interview transcripts. It is possible to see here indications that the approach delivers most contributions, with positive annotations outnumbering the negative annotations. The exceptions were C4 and C5, with negative observations being more frequent. As detailed later, these annotations reflect the fact that participants reported not being able to understand, modify or create platform models (they were only able to reuse them). This was expected, as these participants were students without

[30] The complete novices material and transcripts are available at: https://doi.org/10.5281/zenodo.4728219.

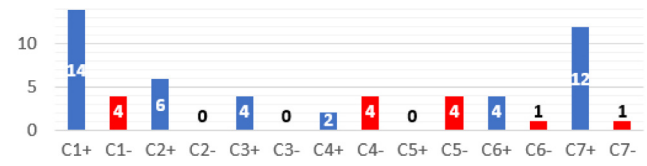

Fig. 7. Number of annotations in favor (**C+**) and against (**C-**) each contribution.

knowledge in mobile or web development, and platform models are essentially a place to put such technical details.

Another interesting observation is that, overall, the novice developers were less critic than the experts, with less negative comments. We believe this is explained by the fact that, in general, less experienced developers are also less capable of performing a deeper analysis of a new technology.

4.3.2. Qualitative analysis

Contribution **C1 (High abstraction level modeling in a single environment)** was the most evident, as shown by the 14 positive annotations. All participants perceived an increase in the abstraction level and the isolation from technical details. Some exemplary comments follow: **N1**: "The GPL is not complex, as it does not have technical details". **N3**: "It is possible to program without knowing the technical details of each platform and thus focus on application logic".

However, there were 4 negative annotations. N3 and N4 pointed out the lack of support for the graphical user interface, and N3 pointed out the lack of resources for testing. N1 also mentioned the lack of programming resources, such as conversion and code completion. And N3 mentioned that the certainty of the code working is only possible in the native IDEs, demanding additional work.

The positive evidence regarding **C2 (Reuse of similar concepts between platforms through modeling)** corresponds to 6 positive annotations in the transcripts, with examples being: **N1**: "I realized it can automate the creation of code for different platforms". **N3**: "I liked the adaptations made by the approach regarding the specifics (technical details) for different entities and platforms".

There were no negative observations regarding **C2**, probably because the participants did not have to deal with technical details in the platform models, only reuse them, and thus everything worked. In comparison with the previous study, this was perceived differently by the experts, who observed that reuse between platforms required extra effort.

Four positive annotations regarding **C3 (Reduced system configuration costs)** were made. Some examples are: **N3**: "... it allows us to really manage which functions will be generated for each platform". **N4**: "You can easily choose where each system function goes". No negative comments were made regarding **C3**.

Contribution **C4 (Inclusion of new platforms with a small impact)** had more negative (4) than positive (2) comments, with the most illustrative examples being: **N1**: "...I don't really know how to use them in other situations, or how to create new global functions". **N3**: "I did not understand how to create a global function".

The participants showed concern and doubts when dealing with global functions. Although they could successfully reuse them, they wondered if they were capable of understanding or creating new ones. An interesting fact is that this was not expected from them, but they reported such concerns anyway. But there were two positive comments regarding the possibility of including new platforms: **N2**: "...it does not limit the user to the platforms supported by the tool manufacturers". **N3**: "It is a nice feature, that enables the user to use and include more platforms".

Evidence regarding **C5 (Extension of the approach through platform models)** is very similar to **C4**. The lack of expertise regarding technical details hindered the participants' confidence in modifying existing platform models: **N1**: "...I understand how to use them... ...but I would not know how to create new functions". **N2**: "I think it is easy to use a global function... ... but I wouldn't know how to create new functions".

Contribution **C6 (Code reuse across domains)** was seen mostly positive. For example: **N2**: "Global functions allow the creation of code for any entity by easy configuration with generic parameters." **N3**: "It is possible to access global functions in any place of the system and reuse the code".

The single negative comment reinforces the lack of confidence in reusing these resources, which is something also mentioned by experts: **N1**: "I don't know if this may create a problem in the reuse of global functions".

C7 (Development and maintenance cost reduction) was the second most identified contribution, with 12 positive annotations, being unanimously cited as a benefit by the participants. Some examples are: **N2**: "What I find most interesting is the reduction of development cost". **N4**: "The great benefit is the agility in development, which end up reducing the costs". The only negative comment was made by **N3**: "But it is necessary to use the native environments for execution and tests, which demands specific knowledge and may reduce productivity".

4.4. Discussion

The first study brought evidence favoring contributions **C1**, **C2**, **C3**, **C4** and **C7**, but this was just a proof-of-concept. The second and third studies brought evidence in favor and against all seven contributions of this approach. The amount of evidence in favor is larger, which indicates that the approach is successful in achieving its goals. We also observed a difference between expert and novice developers. While experts were able to see the benefits in terms of the extension/inclusion of platforms more clearly, novice developers failed to see such benefits, mostly because this requires technical knowledge regarding the platforms.

Another interesting observation was made by a participant. This was not foreseen as an important contribution, but because the third study was conducted with students, it appeared as a side benefit of the approach, as exemplified by the following comment:

N4: "At the same time you learn a new GPL language, you end up learning some Java, Swift, C#. It contributes to the learning of other languages as well."

This is an unexpected benefit, particularly useful for learning. Students can use the approach to see examples of how a piece of abstract code translates to different platforms.

The studies also uncovered eight important limitations. These are summarized as follows, ranked in a decreasing order of severity according to our analysis:

L1. Need to resort to the native language and IDE;

L2. Benefits are only expected in the long term, as global functions need to be fully implemented first;

L3. Lack of support for testing, including tests for validating the system, the platforms and the generated code;

L4. Lack of support for the "View" layer in MVC;

L5. Need to understand a new programming pattern, which may have a steep learning curve;

L6. Lack of trust in the generated code correctness;

L7. Lack of trust in the generated code performance; and

L8. Lack of support for a visual modeling language.

The first limitation is a more fundamental one. Initially, the approach was supposed to support a single environment for creating the software, but in the end it became clear that the native platforms' IDEs have to be used when creating platform implementations. As the time passes and the platforms are implemented and tested, there is less need to resort to the native IDEs. This is also the essence of limitation **L2**.

L3 and **L4** are also important, but we think they can be solved by providing new platforms specifically for these tasks. Regarding **L3**, we conducted a fourth evaluation specifically for this reason, as described later. Regarding **L4**, global functions and details may be used to represent common interface concepts, such as event handling and state management. It might even be possible to create platforms for domain-specific interfaces, based on the fact that many systems from the same domain share similar interfaces. These are the subject of future work.

L5 is also mentioned as an important limitation. The concept of global functions and details requires learning a new programming technique.

L6 and **L7** are the other side of the flexibility provided by the approach. The software engineer has more control over the generated code, but is also responsible for making sure it is correct and adequate.

Finally, **L8** was cited by one of the experts as a possibility for improvement. This limitation was not perceived in practice, however, since the textual language was acknowledged as intuitive and easy to understand.

4.4.1. Threats to validity

There are some aspects that must be discussed regarding possible validity threats in the studies.

The first study was carried out by the researchers, what might have introduced bias. To reduce this bias, no opinions or data related to the learning curve were taken into consideration in this study. Only the feasibility of the approach, its models, and the ability to include a previously unsupported platform were tested in the proof-of-concept.

For the second and third studies, there are also some identified threats. Finding novice developers was easier, as the researchers also work in the academia and are constantly teaching software development subjects to inexperienced developers. But finding experts was a very difficult task, as it required many hours of volunteer work. The researchers resorted to their contacts in academia and industry and it was possible to find five experts that fulfill the requirements (minimum 10 years of experience with mobile and web development and some experience with multiplatform development). However, this sampling was not ideal as the participants knew the researchers and might have provided biased answers. Nevertheless, the amount of negative observations made by the experts was considerable and indicates that the bias might have not been very strong. It also indicates that their level of expertise was adequate enough to allow them to identify real problems, going beyond a superficial analysis.

It was also very difficult to define tasks for the participants. The entire study took an average of 14 h of work from each expert and 17 h from each novice developer. These were not performed uninterruptedly, as the participants worked a small amount of time each day. It took several days to complete all tasks, therefore they were in contact with the approach for a long time. Ideally, the tasks should be longer, so that the participants could have a deeper understanding of all its details. However, this was not possible because their availability for the study was partial.

Still regarding the tasks, we attempted to cover different scenarios to exercise the different contributions of the approach. For example, task T3 asked experts to modify an existing platform to support a new database, and T5 asked experts to include a new platform (Android Wear). In the end, it was possible to collect evidence regarding all contributions. However, no task required an entire system to be developed from scratch. This could probably lead to more identified benefits and limitations but, again, was not possible due to the participants' partial availability.

Another threat is related to a possible external factor. All participants identified productivity gains, but we think that part of these gains might have originated from the *Apache Velocity* templates, and not the approach. Although the templates alone do not promote cross-platform reuse and development, they can greatly reduce programming effort, specially in the CRUD domain. There were some tasks that did not involve the use of such templates, but we believe they might have influenced the participants' responses, who attributed 100% of the productivity gains to the approach, while in reality a fraction of that probably resulted from *Velocity*.

One of the limitations (**L5**) relates to the learning curve regarding the approach, in particular the use of global functions and details. Both experts and novice developers managed to complete the tasks, but they had examples to follow, therefore the learning aspect was not completely addressed.

Finally, the second and third studies had strong subjectivity, in two aspects. Personal opinions are subjective. To reduce this subjectivity, we selected five experts with experience, to increase the confidence that their opinions are not isolated or based on guesswork. The second subjective aspect was the analysis of the participants' opinions, carried out by the researchers. To reduce this threat, the two authors of this paper conducted the analysis separately, and then a consensus was reached after discussion.

4.5. Fourth evaluation: unit testing

This fourth study was conducted by the researchers as a response to observations made by experts and novice developers regarding testing in the platforms (Limitation **L3**). After discussions, we thought this could be done by creating platform models for unit testing in different platforms. In this solution, global functions are created to represent assert functions, present in most unit testing frameworks. Each platform model is responsible for defining how the global assert functions translate to its own platform-specific assert functions, including additional code, such as annotations, if necessary, as global details. Test cases can reuse the global functions and call the assert functions normally.

Listing 9 illustrates the solution. By reusing a global assert function (line 2) and including specific details required by unit test frameworks (line 3), the developer can simply call the assert function (line 21) normally. In this example, an error was deliberately introduced in the "Debit" method (line 16) to cause this test case to fail.

Listing 9: Cross-platform test case using the approach.

```
1  class BankAccountTest{
2   usesGlobal  globalAssertEquals(expected: double,
        actual: double, msg: string): void{}
3   usesGlobalDetails UnitTestAnnotation
4   operation balanceTest(): void{
5    m_customerName: string
6    m_customerName := ''Juliano''
7    beginningBalance: double
8    debitAmount: double
9    expected: double
10   beginningBalance := 11.99
11   debitAmount := 4.55
12   expected := 7.44
```

```
13      account :BankAccount
14      account.setM_customerName(m_customerName)
15      account.setM_balance(beginningBalance)
16      account.Debit(debitAmount)
17      actual : double
18      actual := account.getM_balance()
19      msg: string
20      msg := ‘‘Account not debited correctly’’
21      global globalAssertEquals(expected, actual, msg)
22  }   }
```

After properly configuring code generation for each desired platform, the generated code was successfully executed in the different platforms, as shows Fig. 8. In the figure, the test case from Listing 9 is being executed in iOS and Android, with the same error being accused in both cases.

This evaluation demonstrates that the approach can be used to test application logic in multiple platforms, but there are other issues to be explored, such as testing native resources and integration tests. These are left for future work.

5. Concluding remarks

Cross-platform development solutions have been investigated for a long time, and new research is often arising in the industry and academia. There are many open areas for research, as existing solutions are still not mature enough (Chen et al., 2019).

In this spirit, this paper presents an approach for cross-platform development, where the main idea is to keep development in a single, platform-independent modeling environment, and use generators to produce executable code. The approach takes advantage of similar concepts across platforms and facilitates the inclusion and exchange of platforms used by the system. As a consequence, developers can benefit from reuse provided by code generators and still have control to customize, extend or include new platforms.

The approach targeted seven contributions, selected after an analysis of the solutions available in industry and academia. Four studies were conducted to evaluate them. There were some threats to the validity of the studies, but overall the collected evidence supports the contributions. In particular, the studies demonstrated the ability to extend or modify existing platforms, what is not possible in most existing tools. We also created a new type of device and included it into the approach successfully. In most existing tools, these tasks would be impossible, as the modification of platforms and the inclusion of new platforms depend on the tool vendor, and not the developer. The studies also demonstrated the distribution of functionality among platforms, what is also not a common feature in most existing tools. These features give the approach a holistic view over cross-platform development.

The studies have also shown that it is not currently possible to completely abandon the platforms' native IDEs. This observation goes against the approach's primary goal of using a single environment to create the software. This problem tends to be reduced as time passes and the platforms become more stable, with benefits in the long term.

Future work is being planned to overcome the limitations identified in the studies, as described in Section 4.4. In particular, we plan to experiment with the "View" layer, as it was not considered in the research so far and was pointed out by most participants of the studies as a missing feature.

Some practical improvement possibilities were also identified. One of them is the use of Object Oriented Programming concepts in the platform models, allowing inheritance between platform models and facilitating reuse and system management. One of the experts also suggested a model repository. While it is possible to reuse models in the approach by copying the files, more elaborated ways of reuse could be the subject of future work, so that

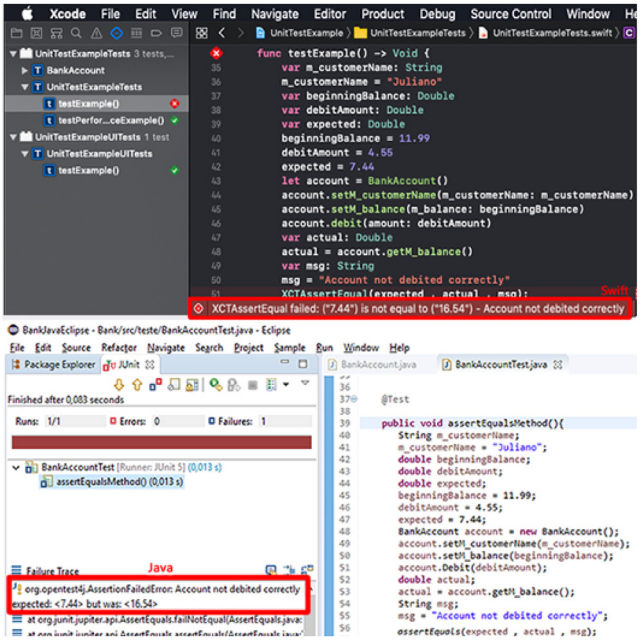

Fig. 8. The same test case executed in iOS and Android.

platform models may be reused within a community. Similarly to code repositories associated with dependency management systems, this effort could help to reduce the initial effort mentioned by experts when using the approach.

Finally, while our approach gives expert developers the possibility to deal with all complexities to each platform, novice developers would be more comfortable if they could be somehow shielded from this complexity, as it is common in most commercial tools. This is also subject of future work.

CRediT authorship contribution statement

J.Z. Blanco: Conceptualization, Methodology, Software, Data curation, Funding acquisition, Project administration, Resources, Writing - original draft, Validation, Investigation. **D. Lucrédio:** Conceptualization, Methodology, Software, Data curation, Funding acquisition, Project administration, Resources, Supervision, Writing - original draft.

Acknowledgment

The research work reported in this paper received financial support from grants #2015/24429-1 and #2017/25343-9 - São Paulo Research Foundation (FAPESP) - Brazil.

References

Acerbis, R., Bongio, A., Butti, S., Brambilla, M., 2015. Model-driven development of cross-platform mobile applications with webratio and ifml. In: Proc. of the Second ACM International Conference on Mobile Software Engineering and Systems. IEEE Press, pp. 170–171.

Behrens, H., 2010. MDSD for the iPhone: developing a domain-specific language and IDE tooling to produce real world applications for mobile devices. In: Proc. of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion. ACM, pp. 123–128.

Benouda, H., Azizi, M., Moussaoui, M., Esbai, R., 2017. Automatic code generation within MDA approach for cross-platform mobiles apps. In: 2017 First International Conference on Embedded & Distributed Systems. EDiS, IEEE, pp. 1–5.

Bucanek, J., 2009. Model-view-controller pattern. In: Learn Objective-C for Java Developers. Springer, pp. 353–402.

Chen, S., Fan, L., Su, T., Ma, L., Liu, Y., Xu, L., 2019. Automated cross-platform GUI code generation for mobile apps. In: 2019 IEEE 1st International Workshop on Artificial Intelligence for Mobile. AI4Mobile, IEEE, pp. 13–16.
Cimino, M.G., Marcelloni, F., 2012. An efficient model-based methodology for developing device-independent mobile applications. J. Syst. Archit. 58 (8), 286–304.
Czarnecki, K., Østerbye, K., Völter, M., 2002. Generative programming. In: European Conference on Object-Oriented Programming. Springer, pp. 15–29.
Dageförde, J.C., Reischmann, T., Majchrzak, T.A., Ernsting, J., 2016. Generating app product lines in a model-driven cross-platform development approach. In: System Sciences (HICSS), 2016 49th Hawaii International Conference on. IEEE, pp. 5803–5812.
Elouali, N., Le Pallec, X., Rouillard, J., Tarby, J.-C., 2014. MIMIC: leveraging sensor-based interactions in multimodal mobile applications. In: CHI'14 Extended Abstracts on Human Factors in Computing Systems. ACM, pp. 2323–2328.
France, R., Rumpe, B., 2007. Model-driven development of complex software: A research roadmap. In: Future of Software Engineering. FOSE '07, pp. 37–54. http://dx.doi.org/10.1109/FOSE.2007.14.
Heitkötter, H., Hanschke, S., Majchrzak, T.A., 2013. Comparing cross-platform development approaches for mobile applications. Web Inf. Syst. Technol. 140, 120–138.
Hoffman, D.L., Novak, T.P., 2018. Consumer and object experience in the internet of things: An assemblage theory approach. J. Consum. Res. 44 (6), 1178–1204.
Inayatullah, M., Azam, F., Anwar, M.W., 2019. Model-based scaffolding code generation for cross-platform applications. In: 2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference. IEMCON, IEEE, pp. 1006–1012.
Jones, C., Jia, X., 2014. The AXIOM model framework: transforming requirements to native code for cross-platform mobile applications. In: Evaluation of Novel Approaches To Software Engineering (ENASE), 2014 International Conference on. IEEE, pp. 1–12.
Litayem, N., Dhupia, B., Rubab, S., 2015. Review of cross-platforms for mobile learning application development. Int. J. Adv. Comput. Sci. Appl. 6 (1).
Majchrzak, T.A., Ernsting, J., Kuchen, H., 2015. Model-driven cross-platform apps: Towards business practicability. In: Proc. of the CAiSE 2015 Forum at the 27th International Conference on Advanced Information Systems Engineering, Vol. 1367. pp. 129–136.
Marcotte, E., 2010. Responsive web design 2010. In: A List Apart. URL: www.alistapart.com/articles/responsive-web-design.
Martinez, M., Lecomte, S., 2017. Towards the quality improvement of cross-platform mobile applications. In: 2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems, MOBILESoft. pp. 184–188.
Min, B.-K., Ko, M., Seo, Y., Kuk, S., Kim, H.S., 2011. A UML metamodel for smart device application modeling based on Windows Phone 7 platform. In: TENCON 2011-2011 IEEE Region 10 Conference. IEEE, pp. 201–205.
Miravet, P., Marin, I., Ortin, F., Rodriguez, J., 2014. Framework for the declarative implementation of native mobile applications. IET Softw. 8 (1), 19–32.
Moreno-Vozmediano, R., Montero, R.S., Huedo, E., Llorente, I.M., 2017. Cross-site virtual network in cloud and fog computing. IEEE Cloud Comput. 4 (2), 46–53.
Perchat, J., Desertot, M., Lecomte, S., 2013. Component based framework to create mobile cross-platform applications. Procedia Comput. Sci. 19, 1004–1011.
Rieger, C., Lucrédio, D., Fortes, R.P.M., Kuchen, H., Dias, F., Duarte, L., 2020. A model-driven approach to cross-platform development of accessible business apps. In: Proceedings of the 35th Annual ACM Symposium on Applied Computing. pp. 984–993.
Sabraoui, A., Abouzahra, A., Afdel, K., Machkour, M., 2019. MDD approach for mobile applications based on DSL. In: 2019 International Conference of Computer Science and Renewable Energies. ICCSRE, IEEE, pp. 1–6.
da Silva, L.P., e Abreu, F.B., 2014. Model-driven gui generation and navigation for android bis apps. In: Model-Driven Engineering and Software Development (MODELSWARD), 2014 2nd International Conference on. IEEE, pp. 400–407.
Taentzer, G., Vaupel, S., 2016. Model-driven development of mobile applications: Towards context-aware apps of high quality. In: PNSE@ Petri Nets. pp. 17–29.
Umuhoza, E., Brambilla, M., Cabot, J., Bongio, A., et al., 2015. Automatic code generation for cross-platform, multi-device mobile apps: Some reflections from an industrial experience. In: Proceedings of the 3rd International Workshop on Mobile Development Lifecycle. ACM, pp. 37–44.
Usman, M., Iqbal, M.Z., Khan, M.U., 2017. A product-line model-driven engineering approach for generating feature-based mobile applications. J. Syst. Softw. 123, 1–32.
Vaupel, S., Taentzer, G., Gerlach, R., Guckert, M., 2018. Model-driven development of mobile applications for Android and iOS supporting role-based app variability. Softw. Syst. Model. 17 (1), 35–63.
Wasserman, A.I., 2010. Software engineering issues for mobile application development. In: Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research, No. 6. FoSER '10, ACM Press, New York, USA, ISBN: 9781450304276, pp. 397–400. http://dx.doi.org/10.1145/1882362.1882443, 978-1-4503. URL: http://portal.acm.org/citation.cfm?doid=1882362.1882443.

Juliano Zanuzzio Blanco is graduated in Data Processing by Americana College of Technology (FATEC-2002), M.Sc. (2009) and PhD (2020) in Computer Science degrees at the Federal University of São Carlos, Brazil. Currently belongs to the faculty of the Federal Institute of São Paulo Campus Piracicaba (IFSP). Has experience in Computer Science, focusing on web and mobile application software development.

Daniel Lucrédio got the Computer engineer (2002) and M.Sc. in Computer Science (2005) degrees at the Federal University of São Carlos, Brazil, and PhD (2009) at University of São Paulo, São Carlos, Brazil, with two doctoral internships: George Mason University (VA, USA) and Microsoft Research (WA, USA). Currently an associate professor at Federal University of São Carlos, Brazil, working mainly with Model-Driven Engineering, Cross-platform software development and Cloud computing.