Towards Pluri-Platform Development: Evaluating a Graphical Model-Driven Approach to App Development Across Device Classes

Christoph Rieger(✉) and Herbert Kuchen

ERCIS, University of Münster, Münster, Germany
{christoph.rieger,kuchen}@uni-muenster.de

Abstract. The domain of mobile apps encompasses a fast-changing ecosystem of platforms and vendors in which new classes of heterogeneous app-enabled devices are emerging. To digitize everyday work routines, business apps are used by many non-technical users. However, designing apps is mostly done according to traditional software development practices, and further complicated by the variability of device capabilities. To empower non-technical users to participate in the creation of supportive apps, graphical domain-specific languages can be used. Consequently, we propose the Münster App Modeling Language (MAML) to specify business apps through graphical building blocks on a high level of abstraction. In contrast to existing process modelling notations, these models can directly be transformed into apps for multiple platforms across different device classes through code generators without the need for manual programming. To evaluate the comprehensibility and usability of MAML's DSL, two studies were performed with software developers, process modellers, and domain experts.

Keywords: Graphical domain-specific language · Model-driven software development · Business app · Cross-platform

1 Introduction

The opportunities of Model-Driven Software Development (MDSD) with regard to increased efficiency and flexibility have been studied extensively in the past years. The use of MDSD techniques is one approach to counteract the variety of platforms, programming languages, and human-interface guidelines found in the domain of mobile business apps. Several approaches have been researched in academic literature, including MD^2 [34], Mobl [26], and AXIOM [30]. Those approaches provide cross-platform development functionalities with one common model for multiple target platforms using textual domain-specific languages (DSLs) to specify apps. Whereas these approaches significantly ease the development of apps and thus also support current trends such as "Bring your own

T. A. Majchrzak et al. (Eds.): *Towards Integrated Web, Mobile, and IoT Technology*, LNBIP 347, pp. 36–66, 2019.
https://doi.org/10.1007/978-3-030-28430-5_3

device" [58], the actual creation of apps is still restricted to users with programming skills [37]. Business apps focus on specific tasks to be accomplished by employees. Therefore, a centralized definition of such processes aligns well with traditional software development practices but may deviate from the end user's needs. Consequently, the introduction of business apps may fall short of improving efficiency. In addition, operating employees have valuable insights into the actual process execution as well as unobvious process exceptions. Giving them a means to shape the software they use in their everyday work routines offers not only the possibility to explicate their tacit knowledge for the development of best practices, but also actively involves them in the evolution of the enterprise. Instead of participating only in early requirements engineering phases of software development, continuously co-designing such systems may increase the adoption of the resulting application and possibly strengthen their identification with the company [19]. Mobile app development can thus benefit from the incorporation of people from all levels of the organization and development tools should be understandable to both programmers and domain experts.

The research company Gartner predicted that more than half of all company-internal mobile apps will be built using codeless tools by 2018 [51]. The general trend towards low-code or codeless development of business apps can be supported by the introduction of graphical notations which are particularly suitable to represent the concepts of a data-driven and process-focused domain. However, current approaches often lack the capacity for holistic app modelling, often operating on a low level of abstraction with visual user interface builders or approaches using view templates (e.g., [22,64]).

In order to advance research in the domain of cross-platform development of mobile apps and investigate opportunities for organizations in a digitized world, this paper presents and evaluates the Münster App Modeling Language (MAML) framework. Rooted in the Eclipse ecosystem, the DSL grammar is defined as an Ecore metamodel, the visual editor is built using the Sirius framework [57], and technologies such as Xtend are used for the code generation of Android, iOS, and Wear OS apps.

Moreover, the high level of abstraction and automatic inference required for simple-to-use app development opens up opportunities for reusing the same notation for heterogeneous target devices. The terms cross-platform or multi-platform development typically denote the creation of applications for multiple platforms *within the same device class*, for example iOS and Android in the smartphone domain – potentially extended to technically similar tablets. However, the development of apps *across device classes* is not yet tackled systematically in academia or practice. To distinguish the additional requirements and challenges introduced when creating applications across heterogeneous devices, we propose the term *pluri-platform development*.

This article greatly extends the paper [46] presented at HICSS 2018[1]. It has been updated to reflect the latest developments, includes new content based on

[1] Please note that verbatim content from the original paper is not explicitly highlighted but for figures and tables already included there.

additional work as well as on the discussions at the conference. Also, it has been amended with a perspective on challenges and opportunities regarding model-driven app development for novel device classes and a study on the suitability of the MAML notation in this context. The remainder of this article is structured as follows: After presenting related work in Sect. 2, MAML's graphical DSL is presented that allows for the visual definition of business apps (Sect. 3). The codeless app creation capabilities are demonstrated using the MAML editor with advanced modelling support and an automated generation of native app source code through a two-step model transformation process. Section 4 discusses the setup and results of two usability studies conducted to demonstrate the potential and intricacies of an integrated app modelling approach for a wide audience of process modellers, domain experts, and programmers. The possibility to extend the approach to heterogeneous devices is covered in Sect. 5. In Sect. 6, the findings and implications of MAML are discussed with regard to model-driven development for heterogeneous app-enabled devices before concluding with a summary and outlook in Sect. 7.

2 Related Work

Different approaches to cross-platform mobile app development have been researched. In general, five approaches can be distinguished [35]. Concerning runtime approaches, *mobile webapps* are browser-run web pages optimized for mobile devices but without native user interface (UI) elements, *hybrid approaches* such as Apache Cordova [3] provide a wrapper to web-based apps that allow for accessing device-specific features through interfaces, and *self-contained runtimes* provide separate engines that mimic core system interfaces in which the app runs. In addition, two generative approaches produce native apps, either by *transpiling* apps between programming languages such as J2ObjC [23] to transform Android-based business logic to Apple's language Objective-C, or *model-driven software development* for transforming a common model to code.

With regard to model-driven development, DSLs are used to model mobile apps on a platform-independent level. According to Langlois et al. [32], DSLs can be classified in textual, graphical, tabular, wizard-based, or domain-specific representations as well as combinations of those. Several frameworks for mobile app development have been developed in the past years, both for scientific and commercial purposes. In the particular domain of business apps – i.e., form-based, data-driven apps interacting with back-end systems [35] – the graphical approach JUSE4Android [15] uses annotated UML diagrams to generate the appearance of and navigation within object graphs, and Vaupel et al. [60] presented an approach focusing around role-driven variants of apps using a visual model representation. Other approaches such as AXIOM [30] and Mobl [26] provide textual DSLs to define business logic, user interaction, and user interface in a common model. An extensive overview of current model-driven frameworks is provided by Umuhoza and Brambilla [59]. However, current approaches mostly rely on a textual specification which limits the active participation of

non-technical users without prior training [65], and graphical approaches are often incapable of covering all structural and behavioural aspects of a mobile app. For generating source code, the work in this paper is based on the Model-Driven Mobile Development (MD^2) framework which also uses a textual DSL for specifying all constituents of a mobile app in a platform-independent manner. After preprocessing the models, native source code is generated for each target platform as described by Majchrzak and Ernsting [34]. This intermediate step is, however, automated and requires no intervention by the user (see Subsect. 3.4).

In contrast to DSLs, several general-purpose modelling notations exist for graphically depicting applications and processes, such as the Unified Modeling Language (UML) with a collection of interrelated standards for software development. The Interaction Flow Modeling Language (IFML) can be used to model user interactions in mobile apps, especially in combination with the mobile-specific elements introduced as extension by Breu et al. [11]. Process workflows can for example be modelled using BPMN [40], Event-Driven Process Chains [1], or flowcharts [27]. However, such notations are often either suitable for generic modelling tasks and remain on a superficial level of detail, or represent rather complex technical notations designed for a target group of programmers [18]. A trade-off is necessary to balance the ease of use for modellers with the richness of technical details for creating functioning apps. Moody [38] has pointed out principles for the cognitive effectiveness of visual notations and subsequent studies have revealed comprehensibility issues through effects such as symbol overload, e.g., for the WebML notation preceding IFML [25]. Examples of technical notations in the domain of mobile applications include a UML extension for distributed systems [54] and a BPMN extension to orchestrate web services [10]. Nevertheless, the approach presented in this work goes beyond pure process modelling. While IFML is closest to the work in this paper regarding the purpose of modelling user interactions, MAML covers both structural (data model and views) and behavioural (business logic and user interaction) aspects.

Lastly, visual programming languages have been created for several domains such as data integration [42] but few approaches focus specifically on mobile apps. RAPPT combines a graphical notation for specifying processes with a textual DSL [8], and AppInventor provides a language of graphical building blocks for programming apps [63]. However, non-technical users are usually ignored in the actual development process. Hence, those visual notations do not exploit the potential of including people with in-depth domain knowledge. Considering commercial frameworks, support for visual development of mobile apps varies significantly. In practice, many recent tools are limited to specific components such as back-end systems or content management, or support particular development phases such as prototyping [43]. Start-ups such as Bizness Apps [9] and Bubble Group [13] aim for more holistic development approaches using configurators and web-based editors. Similarly, development environments have started to provide graphical tools for UI development, enhancing the programmatic specification of views by complementary drag and drop editors [64]. The WebRatio Mobile Platform also supports codeless generation of mobile apps through a combination of

IFML, other UML standards, and custom notations [62]. In contrast, this work focuses on a significantly more abstract and process-centric modelling level as presented in the next section.

3 Münster App Modeling Language

At its core, the MAML framework consists of a graphical modelling notation that is described in the following subsections. Contrary to existing notations, its models contain sufficient information to transform them into fully functional mobile apps. The framework also comprises the necessary development tools to design MAML models in a graphical editor and generate apps without requiring manual programming. The generation process is described in more detail in Subsect. 3.4.

3.1 Language Design Principles

The graphical DSL for MAML is based on five design goals:

Automatic cross-platform app creation: Most important, the whole approach is built around the key concept of codeless app creation. To achieve this, individual models need to be recombined and split according to different roles (see Subsect. 3.4) and transformed into platform-specific source code. As a consequence, models need to encode technical information such as data fields and interrelations between workflow elements in a machine-interpretable way as opposed to an unstructured composition of shapes filled with text.

Domain expert focus: MAML is explicitly designed with a non-technical user in mind. Process modellers as well as domain experts are encouraged to read, modify, and create new models by themselves. The language should, therefore, not resemble technical specification languages drawn from the software engineering domain but instead provide generally understandable visualizations and meaningful abstractions for app-related concepts.

Data-driven process modelling: The basic idea of business apps to focus on data-driven processes determines the level of abstraction chosen for MAML. In contrast to merely providing editors for visual screen composition as replacement for manually programming user interfaces, MAML models represent a substantially higher level of abstraction. Users of the language concentrate on visualizing the sequence of data processing steps and the concrete representation of affected data items is automatically generated using adequate input/output user interface elements.

Modularization: To engage in modelling activities without advanced knowledge of software architectures, appropriate modularization is important to handle the complexity of apps. MAML embraces the aforementioned process-oriented approach by modelling use cases, i.e., a unit of functionality containing a self-contained set of behaviours and interactions performed by the app user [41]. Combining data model, business logic, and visualization in a single model deviates from traditional software engineering practices which, for instance, often

rely on the Model-View-Controller pattern [20]. In accordance with the domain expert focus, the end user is, however, unburdened from this technical implementation issue.

Declarative description: MAML models consist of platform-agnostic elements, declaratively describing *what* activities need to be performed with the data. The concrete representation in the resulting app is deliberately unspecified to account for different capabilities and usage patterns of each targeted mobile platform. The respective code generator can provide sensible defaults for such platform specifics.

3.2 Language Overview

In the following, the key concepts of the MAML DSL are highlighted using the fictitious scenario of a publication management app. A sample process to add a new publication to the system consists of three logical steps: First, the researcher enters some data on the new publication. Then, he can upload the full-text document and optionally revise the corresponding author information. This self-contained set of activities is represented as one model in MAML, the so-called use case, as depicted in Fig. 1.

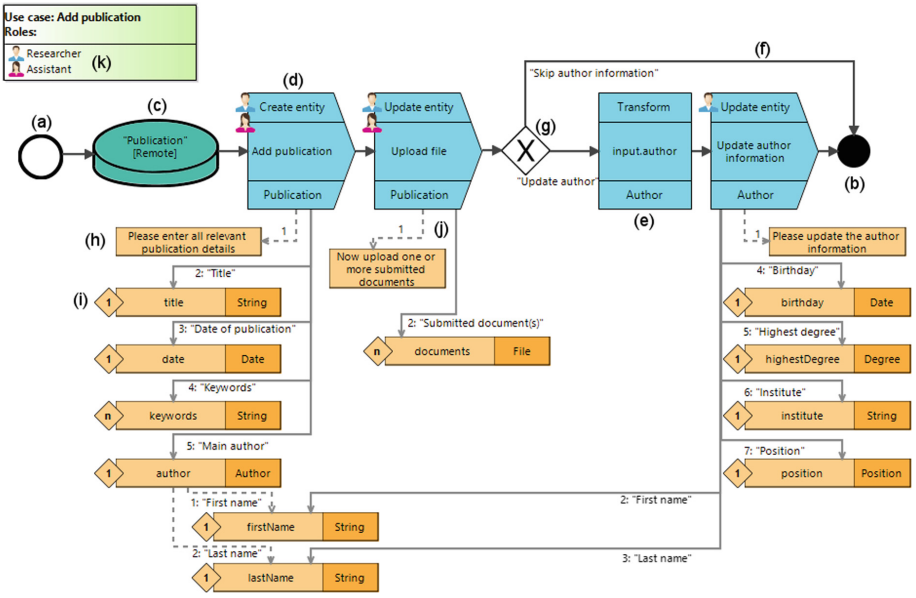

Fig. 1. MAML use case for adding a publication to a review management system [45]

A model consists of a *start event* (labelled with (a) in Fig. 1) and a sequence of process flow elements towards an *end event* (b). A *data source* (c) specifies what type of entity is first used in the process, and whether it is only saved

locally on the mobile device or managed by the remote back-end system. Then, the modeller can choose from predefined *interaction process elements* (d), for example to *create/show/update/delete* an entity, but also to *display messages*, access device sensors such as the *camera*, or *call* a telephone number. Because of the declarative description, no device-specific assumptions can be made on the appearance of such a step. The generator instead provides default representations and functionalities, e.g., display a *select entity* step using a list of all available objects as well as possibilities for searching or filtering. In addition, *automated process elements* (e) represent steps to be performed without user interaction. Those elements provide the minimum amount of technical specificity in order to navigate between the model objects (*transform*), request information from *web services*, or *include* other models to reuse existing use cases.

The order of process steps is established using *process connectors* (f), represented by a default "Continue" button unless specified differently along the connector element. *XOR* (g) elements branch out the process flow based on a manual user action by rendering multiple buttons (see differently labelled connectors in Fig. 1), or automatically by evaluating expressions referring to a property of the considered object.

The lower section of Fig. 1 contains the data linked to each process step. *Labels* (h) provide explanatory text on screen. *Attributes* (i) are modelled as combination of a name, the data type, and the respective cardinality. Data types such as *String, Integer, Float, PhoneNumber, Location*, etc. are already provided but the user can define additional custom types. To further describe custom-defined types, attributes may be nested over multiple levels (e.g., the "author" type in Fig. 1 specifies a first name and last name). In addition, *computed attributes* (not depicted in the example) allow for runtime calculations such as counting or summing up other attribute values.

A suitable UI representation is automatically chosen based on the type of *parameter connector* (j): Dotted arrows signify a reading relationship whereas solid arrows represent a modifying relationship. This refers not only to the manifest representation of attributes displayed either as read-only text or editable input field. The interpretation also applies in a wider sense, e.g., regarding web service calls in which the server "reads" an input parameter and "modifies" information through its response. Each connector also specifies an order of appearance and, for attributes, a human-readable caption derived from the attribute name unless manually specified.

Finally, annotating freely definable *roles* (k) to all interactive process elements allows for the coherent visualization of processes that are performed by more than one person, for example in scenarios such as approval workflows. When a role change occurs, the app automatically saves modified data and users with the subsequent role are informed about the open workflow instance in their app.

3.3 App Modelling

In contrast to other notations, all of the modelling work is performed in a single type of model, mainly by dragging elements from a palette and arranging them on a large canvas. The modelling environment was developed using the Eclipse Sirius framework [57] that was extended with domain-specific validation and guidance to provide advanced modelling support for MAML.

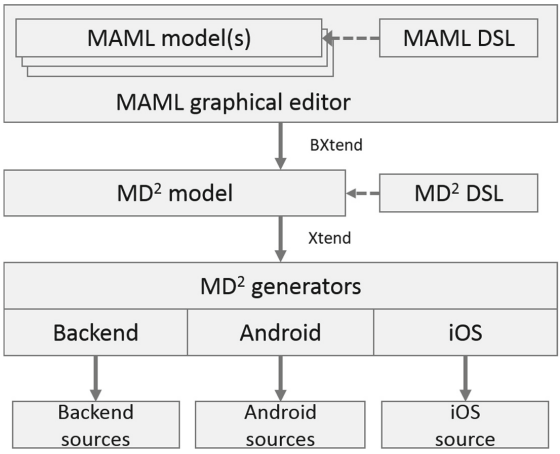

Fig. 2. MAML app generation process (cf. [46])

Modelling only the information displayed in each process step effectively creates a multitude of partial data models for each process step and for each use case as a whole. Also, attributes may be connected to multiple process elements simultaneously, or can be duplicated to different positions to avoid wide-spread connections across the model. An inference mechanism [45] aggregates and validates the complete data model while modelling. During generation, app-internal and back-end data stores are automatically created. As a result, the user does not need to specify a distinct global data model and consistency is automatically checked when models change.

Apart from validation rules to prevent users from modelling syntactically incorrect MAML use cases in the first place, additional validity checks have been implemented in order to detect inconsistencies across use cases (based on the inferred data model) as well as potentially unwanted behaviour (e.g., missing role annotations). Moreover, advanced modelling support attempts to provide guidance and overview to the user. For example, the current data type of a process element (lower label of (d) in Fig. 1) is automatically derived from the preceding elements to improve the user's imagination within the process. Also, suggestions of probable values are provided when adding elements (e.g., known attributes of the originating type when adding UI elements).

3.4 App Generation

Technically, MAML relies on and integrates with the Eclipse Modeling Framework (EMF), for example by specifying the DSL's metamodel as an Ecore model. In order to generate apps, the proposed approach reuses previous work on MD^2 (see Sect. 2). The complete generation process is depicted in Fig. 2. Because of space constraints, the respective transformations are only sketched next.

First, model transformations are applied to transform graphical MAML models to the textual MD^2 representation using the BXtend framework [14] and the Xtend language. Amongst other activities, all separately modelled use cases are recombined, a global data model across all use cases is inferred and explicated, and processes are broken down according to the specified roles. In the subsequent code generation step, previously existing generators in MD^2 create the actual source code for all supported target platforms.

This is, however, not an inherent limitation of the framework. Newly created generators might just as well generate code directly from the MAML model or use interpreted approaches instead of code generation.

It should be noted that this proceeding differs from approaches such as UML's Model Driven Architecture [5] in that the intermediate representation is still a platform-independent representation but with a more technical focus. Optionally, a modeller has the possibility to modify default representations and configure parts of the application in more detail before source code is generated for each platform. Although the tooling around MAML is still in a prototypical state, it currently supports the generation of Android and iOS apps as well as a Java-based server back-end component. Also, a smartwatch generator for Google's Wear OS platform highlights the applicability to further device classes (cf. Sect. 5). The screenshots in Fig. 3 depict the generated Android app views for the first process steps of the MAML model depicted in Fig. 1.

4 Evaluation

As demonstrated, MAML aligns with the goals of automated cross-platform app creation from modular and platform-agnostic app models (cf. Subsect. 3.1). However, the suitability of data-driven process models with regard to the target audience needed to be evaluated in more detail. Therefore, an observational study was performed to assess the utility of the newly developed language. After describing the general setup in Subsect. 4.1, the results on comprehensibility and usability of the graphical DSL are presented.

4.1 Study Setup

The purpose of the study was to assess MAML's claim to be understandable and applicable by users with different backgrounds, in particular including non-technical users. From the variety of methodologies for usability evaluation, observational interviews according to the think-aloud method were selected as empirical approach [21]. Participants were requested to perform realistic tasks with

the system under test and urged to verbalize their actions, questions, problems, and general thoughts while executing these tasks. Due to the novelty of MAML which excludes the possibility of comparative tests, this setup focused on obtaining detailed qualitative feedback on usability issues from a group of potential users.

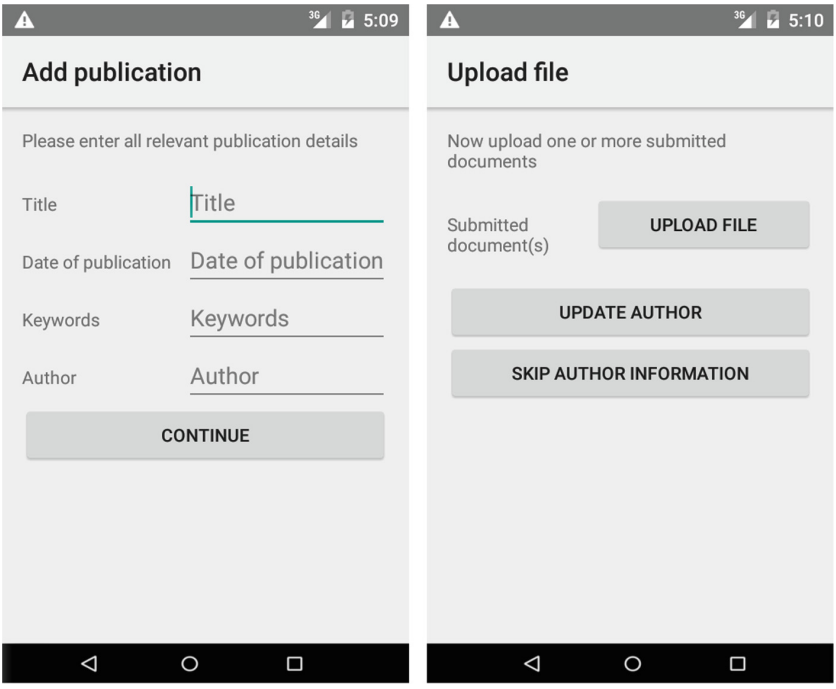

Fig. 3. Exemplary screenshots of generated Android app views [46]

Therefore, 26 individual interviews of around 90 min duration were conducted. An interview consisted of three main parts: First, an online questionnaire had to be filled out in order to collect demographic data, previous knowledge in the domains of programming or modelling, and personal usage of mobile devices. Second, a MAML model and an equivalent IFML model were presented to the participants (in random order to avoid bias) to assess the comprehensibility of such models without prior knowledge or introduction. In addition to the verbal explanations, a short 10-question usability questionnaire was filled out to calculate a score according to the System Usability Scale (SUS) [12] for each notation (cf. Subsect. 4.2). Third, the main part of the interview consisted of four modelling tasks to accomplish using the MAML editor. Finally, the standardized ISONORM questionnaire was used to collect more quantitative feedback, aligned with the seven key requirements of usability according to DIN 9241/110-S [28] (cf. Subsect. 4.3).

To capture the variety of possible usability issues, 71 observational features were identified a priori and structured in six categories of interest: comprehensibility, applying the notation, integration of elements, tool support, effectiveness, and efficiency. In total, over 1500 positive or negative observations were recorded as well as additional usability feedback and proposals for improvement.

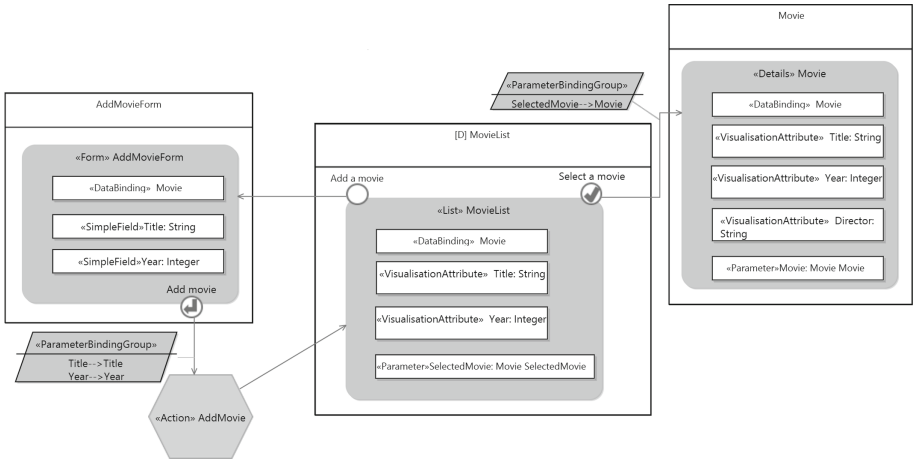

Fig. 4. IFML model to assess the a priori comprehensibility of the notation [46]

Regarding participant selection, 26 potential users in the age range of 20 to 57 years took part in the evaluation. Although they mostly have a university background, technical experience varied widely and none had previous knowledge of IFML or MAML. To further analyse and interpret the results, the participants were categorized in three distinct groups according to their personal background stated in the online questionnaire: 11 software developers have at least medium knowledge in traditional/web/app programming or data modelling, 9 process modellers have at least medium knowledge in process modelling (exceeding their programming skills), and 6 domain experts are experienced in the modelling domain but have no significant technical or process knowledge. Although it is debated whether Virzi's [61] statement of five participants being sufficient to uncover 80% of usability problems in a particular software holds true [56], arguably the selected amount of participants in this study is reasonable with regard to finding the majority of grave usability defects for MAML and generally evaluating the design goals.

For their private use, participants stated an average smartphone usage of 19.2 h per week, out of which 16.3 h are spent on apps. In contrast, tablet use is rather low with 3.5 h (3.2 h for apps), and notebook usage is generally high with 27.5 h but only 4.7 h are spent on apps. For business uses, similar patterns can be observed on total/app-only usage per week on smartphones (5.5h/4.3h), tablets (0.7h/0.2h), and notebooks (18.2h/3.7h). Although this sample is too low for

generalizable insights, the figures indicate a generally high share of app usage on smartphones and tablets compared to the total usage duration, both for personal and business tasks. In addition, with mean values of 1.81/2.12 on a scale between 0 (strongly reduce) and 4 (strongly increase), the participants stated to have no desire of significantly changing their usage volumes of private/business apps.

4.2 Comprehensibility Results

Before actively introducing MAML as modelling tool, the participants should explicate their understanding of a given model without prior knowledge. Comprehensibility is an important characteristic in order to easily communicate app-related concepts via models without the need for extensive training. To compare the results with an existing modelling notation, equivalent IFML (see Fig. 4) and MAML models [47] of a fictitious movie database app were provided with the task to describe the purpose of the overall model and the particular elements. The monochrome models were shown to the participants on paper in randomized order to avoid bias from priming effects [7] and potential influences from a particular software environment.

After each model, participants were asked to answer the SUS questionnaire for the particular notation. This questionnaire has been applied in many contexts since its development in 1986 and can be seen as easy, yet effective, test to determine usability characteristics. Each participant answers ten questions using a five-point Likert-type scale between strong disagreement and strong agreement, which is later converted and scaled to a [0;100] interval according to Brooke [12]. The participants' scores for both languages and the respective standard deviations are depicted in Table 1.

Table 1. System usability scores for IFML and MAML

SUS ratings	IFML	MAML
All participants	52.79 ($\sigma = 23.0$)	66.83 ($\sigma = 15.6$)
Software developers	45.91 ($\sigma = 23.6$)	64.09 ($\sigma = 17.3$)
Process modellers	64.17 ($\sigma = 19.0$)	69.44 ($\sigma = 12.0$)
Domain experts	48.33 ($\sigma = 24.5$)	67.92 ($\sigma = 18.7$)

However, it should be noted that the results do not represent percentage values. Instead, an adjective rating scale was proposed by Bangor et al. [6] to interpret the results as depicted in Fig. 5. The results show that MAML's scores are superior overall as well as for all three groups of participants. In addition, the consistency of scores across all groups supports the design goal of creating a notation which is well understandable for users with different backgrounds. Particularly, domain experts without technical experience expressed a drastic difference in comprehensibility of almost 20 points.

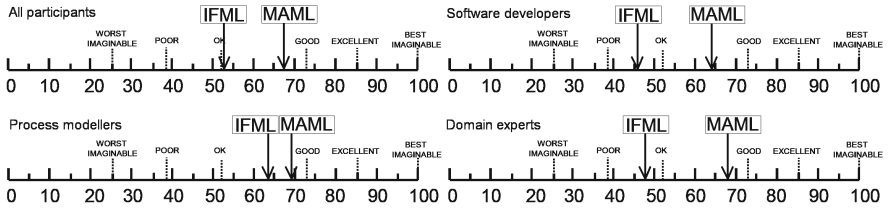

Fig. 5. SUS ratings for IFML and MAML [46]

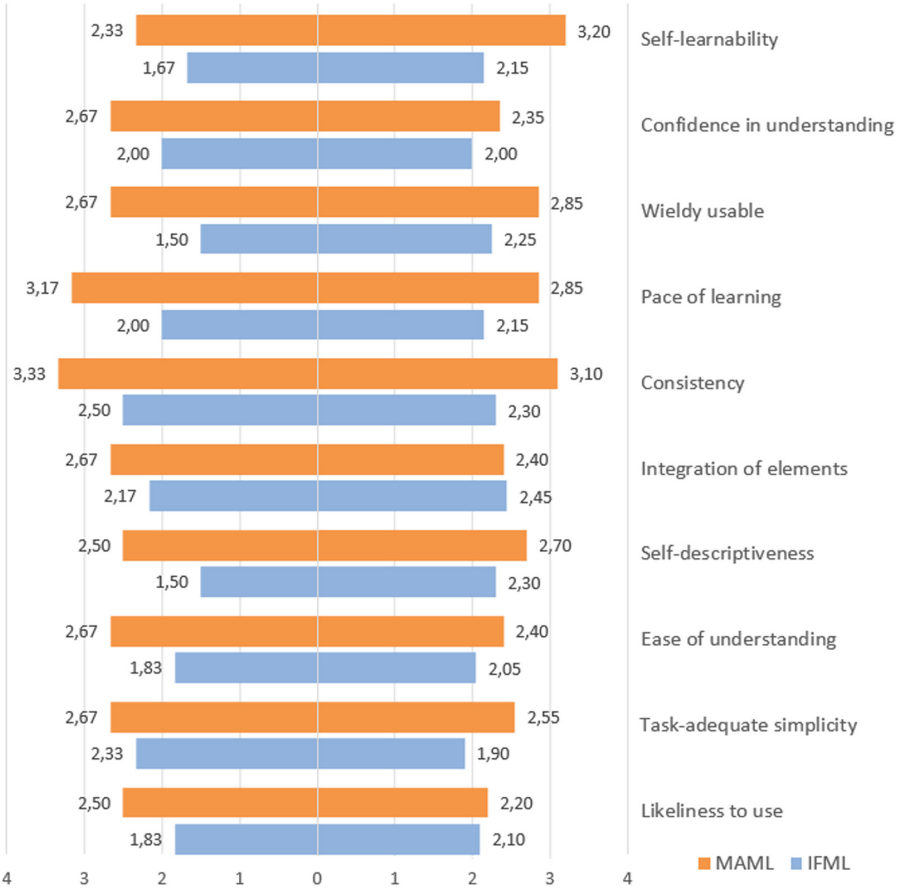

Fig. 6. SUS answers for domain experts (left) and technical users (right) [48]

Especially the distinction between domain experts and technical users (developers and process modellers together) is of interest to evaluate the design goal of MAML to be comprehensible for different user groups. Figure 6 breaks down the answers to the 10 questions of the SUS questionnaire (rescaled to a [0;4] interval; 4 denoting strong acceptance). With one exception, responses for MAML are

higher than for the technical IFML notation. Moreover, domain experts reacted significantly more positively when assessing the MAML notation as being wieldy usable (+1.17 compared to IFML), fast to learn (+1.17), and self-descriptive (+1.00). Consequently, the understandability and general applicability of the notation is in the focus of domain experts, which aligns well with the intention to use MAML for communicating with potential end users and include them in the development process. The strongest deviations for technical users, in contrast, can be seen in questions regarding self-learnability (+1.05), perceived consistency (+0.80), and pace of learning (+0.70). Conforming with their technical background, these aspects emphasize the correct application of the notation which apparently is perceived as positive in MAML, too.

Considering also the qualitative observations, some interesting insights can be gained. According to the questionnaire results, most of the criticism is related to the categories "easy to understand" and "confidence in the notation". IFML's approach of visually hinting at the outcome through the order of elements and their composition in screen-like boxes was often noted as positive and slightly more intuitive compared to MAML. This argument is not unexpected as the level of abstraction was designed to be higher than a pure visual equivalent of programming activities. Also, the notation is not limited to the few types of mobile devices known by a participant, e.g., smartwatches and smartphones exhibit very different interface and interaction characteristics. Therefore, a fully screen-oriented approach generally contradicts the desired platform-independent design of MAML. However, this is valuable feedback for the future, e.g., improving modelling support by using an additional simulator component to preview the outcome while modelling.

Surprisingly, IFML scores were worst for the group of software developers, although they have knowledge of other UML concepts and diagrams. Despite this apparent familiarity, reasons for the negative assessment of IFML can be found in the amount of "technical clutter", e.g., regarding parameter and data bindings, as well as perceived redundancies and inconsistencies. In contrast, 86% of these participants highlight the clarity of MAML regarding the composition of individual models and 88% are able to sketch a possible appearance of the final app result based on the abstract process specification.

Overall, three in four participants can also transfer knowledge from other modelling notations, e.g., to interpret elements such as data sources. All participants within the process modeller group immediately recognize analogies from other graphical notations such as BPMN, and understand the process-related concepts of MAML. Whereas elements such as data sources (understood by 75% of all participants) and nested attribute structures (83%) are interpreted correctly on an abstract level, comprehensibility drops with regard to technical aspects, e.g., data types (57%) or connector types (43%).

Finally, domain experts also have difficulties to understand the technical aspects of MAML without previous introduction. Although concepts such as cardinalities (0%), data types (25%), and nested object structures (67%) are not initially understood and ignored, all participants in this group are still able to

visualize the process steps and main actions of the model. As described in Subsect. 3.1, further reducing these technical aspects constrains the possibilities to generate code from the model. Some suggestions exist to improve readability, e.g., replacing the textual data type names with visualizations. Nevertheless, MAML is comparatively well understandable. Curiously enough, the sample IFML model is often perceived as being a more detailed technical representation of MAML instead of a notation with equivalent expressiveness.

To sum up, MAML models are favoured by participants from all groups, despite differences in personal background and technical experience. This part of the study is not supposed to discredit IFML but emphasizes their different foci: Whereas IFML covers an extensive set of features and integrates into the UML ecosystem, it is originally designed as generic notation for modelling user interactions and targeted at technical users. In contrast, the study confirms MAML's design principle of an understandable DSL for the purpose of mobile app modelling.

4.3 Usability Results

In addition to the language's comprehensibility, a major part of the study evaluated the actual creation of models by the participants using the developed graphical editor. After a brief ten-minute introduction of the language concepts and the editor environment, four tasks were presented that cover many of MAML's features and concepts. In the hands-on context of a library app (cf. supplementary online material [47]), a first simple model to add a new book to the library requires the combination of core features such as process elements and attributes. Second, participants should model how to borrow a book based on screenshots of the resulting app. This requires more interaction element types, a case distinction, and complex attributes. Third, modelling a summary of charges includes a web service call, exception handling, and calculations. Fourth, a partial model in a multi-role context needed to be altered.

The final evaluation was performed using the ISONORM questionnaire in order to assess the usability according to the ISO 9241-110 standard [28]. 35 questions with a scale between −3 and 3 cover the seven criteria of usability as presented in Table 2. Again, MAML achieves positive results for every criterion, both for the participant subgroups and in total. Taking the interview observations into account for qualitative feedback, these figures can be evaluated in more detail.

Regarding the *suitability for the task*, observations on the effectiveness and efficiency of the notation show that handling models in the editor is achieved without major problems. 94% of the participants themselves noticed a fast familiarization with the notation, although domain experts are generally more wary when using the software. The deliberately chosen high level of abstraction manifests in 37% of participants describing this approach as uncommon or astonishing (see also Sect. 6). Nevertheless, 67% of the participants state to have an understanding of the resulting app while modelling.

Table 2. ISONORM usability questionnaire results for MAML.

Criterion	All participants	Software developers	Process modellers	Domain experts
Suitability for the task	1.63 ($\sigma = 1.04$)	1.36 ($\sigma = 1.13$)	1.62 ($\sigma = 1.12$)	2.13 ($\sigma = 0.62$)
Self-descriptiveness	0.51 ($\sigma = 0.73$)	0.62 ($\sigma = 0.62$)	0.38 ($\sigma = 1.02$)	0.50 ($\sigma = 0.41$)
Controllability	2.10 ($\sigma = 0.83$)	2.20 ($\sigma = 0.63$)	2.02 ($\sigma = 0.63$)	2.03 ($\sigma = 1.41$)
Conformity with user expectations	1.78 ($\sigma = 0.52$)	1.85 ($\sigma = 0.47$)	1.64 ($\sigma = 0.47$)	1.87 ($\sigma = 0.70$)
Error tolerance	0.92 ($\sigma = 0.96$)	0.89 ($\sigma = 0.63$)	1.11 ($\sigma = 0.81$)	0.70 ($\sigma = 1.63$)
Suitability for individualisation	1.20 ($\sigma = 0.90$)	1.04 ($\sigma = 1.05$)	1.42 ($\sigma = 1.02$)	1.17 ($\sigma = 0.27$)
Suitability for learning	1.83 ($\sigma = 0.67$)	2.02 ($\sigma = 0.54$)	1.69 ($\sigma = 0.66$)	1.70 ($\sigma = 0.90$)
Overall score	1.43 ($\sigma = 0.49$)	1.43 ($\sigma = 0.46$)	1.41 ($\sigma = 0.53$)	1.44 ($\sigma = 0.59$)

Self-descriptiveness refers to comprehension issues but additionally deals with the correct integration of different elements while modelling. For example, the concept of user roles was introduced to the participants but not the assignment in models. Still, 86% of them intuitively drag and drop role icons on process elements correctly. Furthermore, process exceptions were not explained at all in the introduction but 71% of the participants applied the "error event" element correctly without help. Self-descriptiveness is, however, more limited when dealing with technical issues. Side effects of transitive attributes are only recognized by 43% of process modellers and 25% of domain experts. Model validation or additional modelling support is needed in order to guide the users towards semantically correct models. Similarly, the complexity of modelling web service responses within the use case's data flow poses challenges to 44% of the participants.

The very positive responses for the *controllability* criterion can be explained by the simplistic design of MAML and its tools. All modelling activities are performed in a single model instead of switching between multiple perspectives. In contrast to other notations, all of the modelling work is performed in a single type of model, mainly by dragging elements from a palette and arranging them on a large canvas. Many participants utter remarks such as "the editor does not evoke the impression of a complex tool". In parts, this impression can be attributed to sophisticated modelling support, including live data model inference when connecting elements in the model, validation rules, and suggestions for available data types.

Related to the clarity of possible user actions, the *conformity with user expectations* is also clearly positive. Despite occasional performance issues caused by the prototypical nature of the tools, a consistent handling of the program is confirmed by the participants. Although aspects such as the direction of parameter connections may be interpreted differently (e.g., either a sum refers to attributes or attributes are incoming arguments to the sum function), the consistent use of concepts throughout the notation is easily internalized by the participants.

Regarding *error tolerance* and *suitability for individualisation*, scores are moderate but the prototype was not yet particularly optimized for production-ready stability or performance. Also, an individual appearance was not intended, thus providing only basic capabilities such as resizing and repositioning components. Whereas the editor is very permissive with regard to the order of modelling activities, adding invalid model elements is mostly avoided by syntactic and semantic validity checks, e.g., which elements are valid end points of a connector. Participants appreciate the support of not being able to model invalid constellations. However, criticism arises from disallowing actions without further feedback on why a specific action is invalid. The modelling environment Sirius is currently not able to provide this information, yet users might benefit more from such dynamic explanations than from traditional help pages.

Finally, *suitability for learning* can be demonstrated best using quotes such as MAML being judged as "a really practical approach", and participants having "fun to experiment with different elements" or being "surprised about what I am actually able to achieve". Using the graphical approach, users can express their ideas and apply concepts consistently to different elements. As mentioned above, many unknown features such as roles or web service interactions can be learned using known drag and drop patterns or read/modify relationships.

5 Towards Pluri-Platform Development

The term *cross-platform* as well as actual development frameworks in academia and practice are usually limited to smartphones and sometimes – yet not always – technically similar tablets. Thus, they ignore the differing requirements and capabilities within the variety of novel devices and platforms reaching the mainstream consumer market in the near future. Extending the boundaries of current cross-platform development approaches requires a new scope of target devices which can be subsumed under the term *app-enabled* devices. Following the definition in [50], an app-enabled device can be described as being extensible with software that comes in small, interchangeable pieces, which are usually provided by third parties unrelated to the hardware vendor or platform manufacturer, and increase the versatility of the device after its introduction. Although these devices are typically portable or wearable and therefore related to the term *mobile computing*, there are further devices classes with the ability to run apps (e.g., smart TVs).

However, new challenges arise when extending the idea of cross-platform development to app-enabled device classes. In particular, not all approaches mentioned in Sect. 2 are generally capable for this extension as described in the following.

5.1 Challenges

From the development and usage perspectives on app development, specific challenges can be identified related to app development across device classes and which can be grouped into four main categories.

Output Heterogeneity: The user interface of upcoming device classes enables more flexible and intuitive ways of device interaction compared to the prevalent focus on medium screen sizes between 4″ and 10″. By design, graphical output is very limited on wearables. On the other hand, smart TVs provide large-scale screens beyond 20″. Also, devices can use new techniques for presenting information, e.g., auditive output by smart virtual assistants, or projection through augmented reality (for example using the wind shield in vehicles). In addition, even for screen-based devices the variability of output increases because of new screen designs with drastically differing pixel density, aspect ratios, and form factors (e.g., round smartwatches) [50]. Techniques from the field of adaptive user interfaces may be used to tackle these issues. To achieve this degree of adaptability, specifying user interfaces needs to evolve from a screen-oriented specification of explicitly positioned widgets to a higher level of abstraction which can use semantic information to transform the content to a particular representation.

User Input Heterogeneity: The device interfaces for entering information by the user also evolve and will use a wider spectrum of possible techniques for user input [50]. This ranges from pushing buttons attached to the device, using remote controls, directing pointing devices for graphical user interfaces, tapping on touch screens, and using auxiliary devices (e.g., stylus pens) to hands-free interactions via gestures, voice, or even neural interfaces. To complicate matters, a single device may provide multiple input alternatives for convenience and especially new device classes are often experimenting with different interactions patterns. Again, this complexity calls for a higher level of abstraction when specifying apps by decoupling actual input events from the intended actions of the user interaction.

Device Class Capabilities: The variability of hardware and software across device classes is also apparent besides the user interface. For example, the miniaturization in wearable devices negatively impacts the computational power and battery capacity. Complex computations may therefore be offloaded to potential companion devices or provided through edge/cloud computing [44]. Sensing capabilities can vary both within and across device classes. In addition, platform operating systems provide different levels of device functionality access and app interoperability, e.g., regarding security issues in vehicles. To avoid the problem of developing for the least common denominator of all targeted devices, suitable replacements for unavailable sensors need to be provided. For example, automatic location detection via GPS sensors can have fallback solutions such as address lookup or manual selection on a map.

Multi-device Interaction: Whereas cross-platform approaches often provide self-contained apps with the same functionality for different users (it is fairly uncommon to own multiple smartphones with different platforms), users increasingly own multiple devices of different device classes and aim for interoperable solutions within their ecosystem. This complexity of multi-device interactions for a single user might occur *sequentially* when a user switches to a different device

depending on the usage context or user preferences (e.g., reading notifications on a smartwatch and typing the response on a smartphone for convenience). Moreover, a *concurrent* usage of multiple devices for the same task is possible, for instance in a second screening scenario in which one device provides additional information or input/output capabilities for controlling another device [39]. In both cases, fast and reliable synchronization of content is essential in order to seamlessly switch between multiple devices.

5.2 Towards Pluri-Platform Development

To emphasize the difference in scope and the respective solution approaches compared to traditional cross-platform development, we propose the term *pluri-platform* development to signify the creation of apps *across* device classes, in contrast to multi-/cross-platform development for several platforms *within* one class of mostly homogeneous devices. Pluri-platform development can, therefore, be understood as an umbrella term for different approaches aiming to bridge the gap between multiple device classes by tackling the challenges of heterogeneous input and output mechanisms, device capabilities, and multi-device interactions. In contrast to cross-platform development, the focus lies on simplification of app creation not just with regard to the representation of user interfaces but also the integration with platform-specific usage patterns and the interaction within a multi-device context. The related research fields of adaptive user interfaces and context-aware interfaces thus only account for a subset of the required solutions to achieve pluri-platform development.

Considering previous literature in this domain, very few works explicitly deal with app development *spanning multiple device classes*, indicating that app development beyond smartphones is not yet approached systematically but on a case-by-case basis. Cross-platform overview papers such as [29] typically focus on a single category of devices and apply a very narrow notion of mobile devices. [50] provides the only classification that includes novel device classes. Few papers provide a *technical* perspective on apps spanning multiple device classes. Singh and Buford [55] describe a cross-device team communication use case for desktop, smartphones, and wearables, and Esakia et al. [17] performed research on the interaction between the Pebble smartwatch and smartphones in computer science courses. In the context of Web-of-Things devices, Koren and Klamma [31] propose a middleware approach to integrate data and heterogeneous UI, and Alulema et al. [2] propose a DSL for bridging the presentation layer of heterogeneous devices in combination with web services for incorporating business logic.

With regard to commercial cross-platform products, Xamarin [64] and CocoonJS [33] provide Wear OS support to some extent. Whereas several other frameworks claim to support wearables, this usually only refers to accessing data by the main smartphone application or displaying notifications on already coupled devices.

Together with the increase in devices, new software platforms have appeared, some of which are either related to established operating systems (OS) for other

device classes or are newly designed to run on multiple heterogeneous devices. Examples include Android/Wear OS, watchOS, and Tizen. Although these platforms ease the development of apps (e.g., reusing code and libraries), subtle differences exist in the available functionality and general cross-platform challenges remain.

5.3 Applicability of Existing Cross-Platform Approaches for Pluri-Platform Development

Different instantiations and practical frameworks may be conceived which extend the approaches to cross-platform development presented in Sect. 2.

A plethora of literature exists in the context of cross- or multi-platform development. Classifications such as in [16] and [36] have identified five main approaches to multi-platform app development which are varyingly suited to the specific challenges of pluri-platform development. With regard to runtime-based approaches, *mobile webapps* – including recently proposed Progressive Web Apps – are mobile-optimized web pages that are accessed using the device's browser and relatively easy to develop using web technologies. However, most novel device types, e.g., the major smartwatch platforms watchOS [4] and Wear OS by Google [24], do not provide WebView components or browser engines that allow for the execution of JavaScript code. Consequently, this approach cannot be used for pluri-platform development targeting a broader range of devices.

Hybrid apps are developed similarly using web technologies but are encapsulated in a wrapper component that enables access to device hardware and OS functionality through an API. Although they are distributed as app packages via marketplaces, hybrid apps rely on the same technology and can neither be used for pluri-platform development.

In contrast, a *self-contained runtime* does not depend on the device's browser engine but uses platform-specific libraries provided by the framework developer in order to use native functionality. Of the runtime environment approaches, this is the only one that can be used for pluri-platform development. Although usually based on custom scripting languages, a runtime can also be used as a replacement for inexistent platform functionality. As an example, CocoonJS [33] recreated a restricted WebView engine and, therefore, supports the development of JavaScript-based apps also for the Wear OS platform. However, devices need sufficient computing power to execute the runtime on top of the actual operating system. Also, synergies with regard to user input/output and available hardware/software functionality across heterogeneous devices are dependent on the runtime's API.

Considering generative approaches to cross-platform development, *model-driven software development* has several advantages as it uses textual or graphical models as main artefacts to develop apps and then generates native source code from this platform-neutral specification. Referring to domain-specific concepts allows for a high level of abstraction, for example circumventing issues such as input and output heterogeneity using declarative notations. Arbitrary platforms can be supported by developing respective generators which

implement a suitable mapping from descriptive models to native platform-specific implementations.

Finally, *transpiling* approaches use existing application code and transform it into different programming languages. Pluri-platform development using this approach is technically possible as the result is also native code. However, there is more to app development than just the technical equivalence of code, which also explains the low adoption of this approach by current cross-platform frameworks. For instance, user interfaces behave drastically differently across different device classes, and substantial transformations would be required to identify the contextual patterns from the low-level implementation. It is therefore unlikely that this approach provides a suitable means for pluri-platform development beyond reusing individual components such as business logic.

To sum up, only self-contained runtimes and model-driven approaches are candidates for pluri-platform development, of which the latter additionally benefits from the transformation of domain abstractions to platform-specific implementations.

5.4 Evaluation of MAML in a Pluri-Platform Context

The MAML notation condenses the development of apps to a sequence of data manipulation activities. Conceptually, the platform-agnostic nature and high level of abstraction allow for a wider applicability beyond just smartphone platforms. From the variety of novel app-enabled device classes [50], smartwatches have so far become most prevalent on the consumer market which offers a multitude of devices and several vendors promoting new platforms. Today's situation resembles the early experimental years after the introduction of the iPhone in 2007 and development using adequate abstractions is needed. Although a smartwatch typically has a touch screen, the screen dimension as well as the user input mechanisms, sensing capabilities, and usage patterns differ from smartphones. Ideally, pluri-platform development approaches can bridge the gap between app development not only for different smartwatches but integrate with existing ecosystems of current app-enabled devices. This is especially beneficial in a multi-device context in which one user owns several devices and can use platform-adapted apps depending on personal preferences or usage contexts.

To investigate the practical opportunities and challenges of pluri-platform development, we developed a new code generator for the Wear OS platform by Google (formerly Android Wear) [24] which supports the creation of stand-alone apps for respective smartwatches. Consequently, the model-driven foundation of our framework now allows for the combined generation of smartphone and smartwatch source code using the same MAML models as input. More details on the required transformations to represent the desired content on a smartwatch are presented in [49]. Yet, in this work we want to focus on the usability of the notation with regard to the specification of apps across device classes.

Therefore, a second study was conducted in order to validate the previous results in the established smartphone domain and gain insights on the suitability towards other app-enabled device classes. The study was conducted with 23

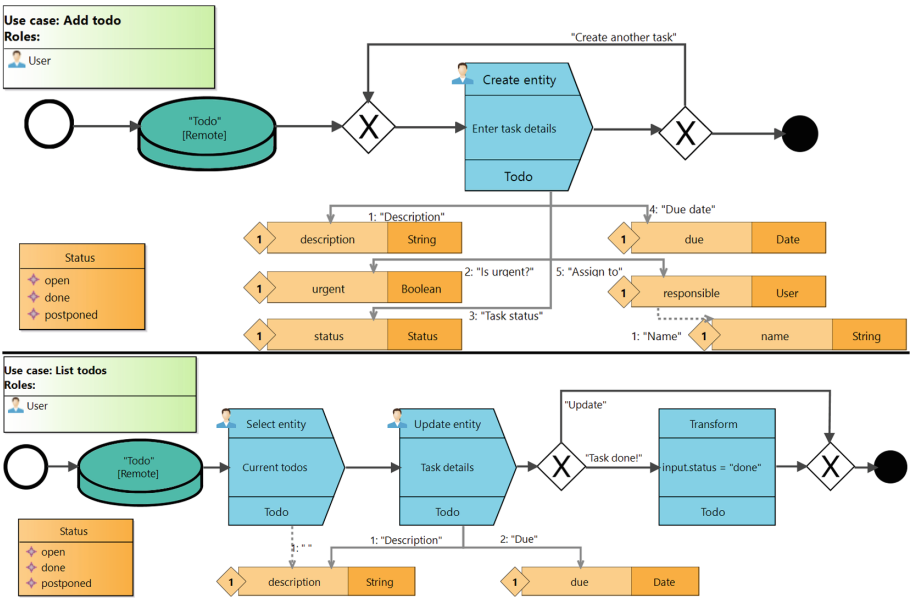

Fig. 7. Use cases for adding and displaying items in a to-do management system [45]

students from a course on advanced concepts in software engineering of an information systems master's program. Whereas designing applications using MDSD techniques is part of the course contents and knowledge of process modelling notations can be presumed, no previous experience with app development was expected in order to avoid a bias towards existing frameworks or approaches. This is supported by the average responses regarding experience in the development of web apps (3.26), hybrid apps (4.35), and native apps (4.30) on a 5-point Likert scale.

Using a simple to-do management scenario depicted in Fig. 7, a 5-min introduction to the MAML notation was given to explain the two processes of creating a new to-do item in the system and displaying the full list of to-dos with the possibility to update items and ticking off the task. Subsequently, participants were asked to express their conceptions of the resulting apps by sketching smartphone and smartwatch user interfaces complying with these use cases.

Interestingly, 64% of the participants intuitively chose a square representation for the smartwatch screen, which reflects the publicity of the Apple watch. Also, a variety of interaction patterns could be derived from the sketches, for example the representation of repetitive elements as a vertical scrollable list (65%) in contrast to 17% using a horizontal arrangement. From the sketches that hint towards navigation patterns, the master-detail pattern of the "list todos" use case of Fig. 7 was conceived either via tapping on the element (42%), using an edit button (33%), pressing a hardware button such as the watch crown (8%), swiping to the right hand side (8%), or using a voice command (8%).

As regards the creation of new entities, 35% of the participants imagined a scrollable view containing all attributes, whereas 30% decided for separate input steps for each attribute, 9% utilized the available screen dimensions and distributed the attributes across multiple views with more than one attribute on each. In addition, 30% allowed the unstructured input of data via voice interface and 26% allowed voice inputs per attribute (the total above 100% results from multiple alternatives combined in the same sketches). It can also be said that a common perception of navigation within a smartphone app has not been established so far: From the identifiable navigation patterns, 53% relied on buttons to continue through the creation process whereas horizontal or vertical swipe gestures were depicted in 20% and 13%, respectively, and 14% decided to use one or multiple hardware buttons for navigation.

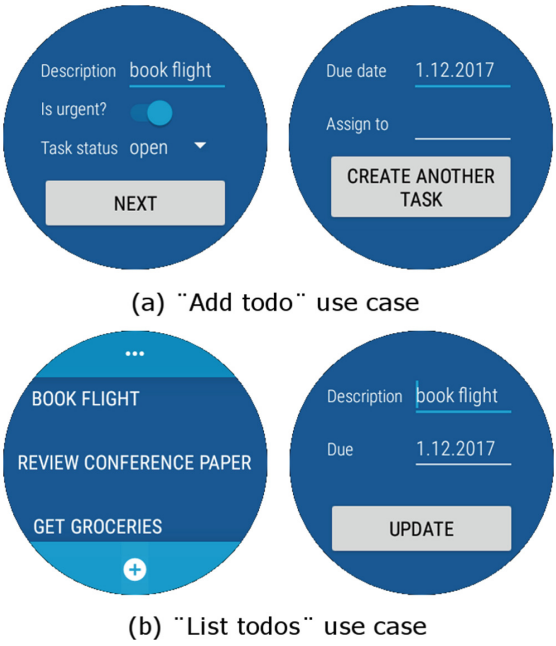

Fig. 8. Generated Wear OS app for the system modelled in Fig. 7 (cf. [49])

The standardized SUS questionnaire was used to triangulate the results with the initial study presented in Sect. 4. The resulting score of 66.85 ($\sigma = 12.9$) aligns very well with the figures depicted in Table 1 for software developers and process modellers and reinforces the validity of the previous study. Upon showing the generated app result depicted in Fig. 8, the participants were asked about their opinion on the smartwatch outcome (using again a 5-point Likert scale). The participants agreed (2.04) that the generated smartwatch app suitably represents the process depicted in the MAML model. Furthermore, they supported

the statement (2.39) that the resulting app is functional with regard to the to-do management scenario. The visual appearance of the smartwatch was rated merely with 3.3 which can be explained by the generic transformations and assumptions derived from the abstract process model. Also, the prototypical nature of our generator needs more refinements to choose suitable representations.

Regarding the combined generation of apps for smartwatch and smartphone from the same model, the participants did not feel that the common notation makes app development unnecessarily complex (3.35) and tended to agree that having one notation for both app representations accelerates app development (2.48). When asked about specific durations, the students estimated the required time to build the MAML models with 50 min on average, compared to a mean value of 27.3 h when programming the application natively or with cross-platform programming frameworks. Though the actual development was not performed in this study, these estimates underline the possible economic impact of MDSD to reduce the effort for creating specific applications and thus achieve a faster time to market for new apps or app updates.

6 Discussion

In this section, key findings of the proposed MAML framework and subsequent evaluation are discussed with regard to the design objectives and general implications on model-driven software development for mobile applications across device classes. Regarding the principle of data-driven process modelling, using process flows in a graphical notation has shown to be a suitable approach for declaratively designing business apps. Graphical DSLs can also simplify modelling activities for the users of other domains, especially those that benefit from a visual composition of elements such as graph structures. Particularly for MAML, the chosen level of abstraction allows for a much wider usage compared to low-level graphical screen design: Besides the actual app product, models can be used to discuss and communicate small-scale business processes in a more comprehensive way than BPMN or similar process notations through combined modelling of process flows and data structures. In contrast to alternative codeless app development approaches focused on the graphical configuration of UI elements, users do not get distracted by the eventual position of elements on screen but can focus on the task to be accomplished. Moreover, the DSL is platform-agnostic and can thus be used to describe apps for a large variety of mobile devices. Apart from smartphones and tablets, generators for novel device types such as smartwatches or smart glasses may be created in the future based on the same input models.

Second, the challenge of developing a machine-interpretable notation that is understandable both for technical and non-technical users is a balancing act, but the interview observations and consistent scores in the evaluation indicate this design goal was reached. The most significant differences in the participants' modelling results are related to technical accuracy, mostly because of (missing) knowledge about programming or process abstractions. As such issues

not always manifest as modelling errors but often happen through oversights, preventing them while keeping a certain joy of use is only achievable using a combined approach: The notation itself should be permissive instead of overly formal. Moreover, clarity (e.g., wording of UI elements) and simplicity of the DSL contribute to manageable models. Most important, however, is the extensive use of modelling support for different levels of experience. Novice users learn from hints (e.g., hover texts and error explanations) whereas advanced users can benefit from domain-specific validation rules and optional perspectives to preview results of model changes. Particularly for MAML, advanced modelling support is achieved by interpreting the models and inferring a global object structure from a variety of partial data models as described in [45]. Consequently, this feature allows for dynamically generated suggestions such as available data types, implicit reactions such as forbidding illegitimate element connections, and validation of conflicting data types and cardinalities. In general, a model-driven approach with advanced modelling support enables the active involvement of business experts in software development processes and can be regarded as major influencing factor for a successful integration of non-programmers.

Finally, the choice of mixing data model, business logic, and view details in a single model deviates from traditional software engineering practices in order to ease the modelling process for non-technical users. This does not mean that we recommend MAML for all process-oriented modelling tasks. Large business processes are just too complex to be jointly expressed with all data objects in a single model. However, mobile apps with small-scale tasks and processes are well suited to this kind of integrated modelling approach. The evaluation has shown that users appreciate the simplicity of the editor without switching between multiple interrelated models, a major distinction from related approaches to graphical mobile app development. Possibly related to the aforementioned modelling support, not even programmers miss the two-step approach of first specifying a global data model and then separately defining the respective processes. Nevertheless, as potential future extension, an optional view of the inferred data model may be interesting for them to check the modelling result before generation. Similarly, two non-technical users stated the wish for a preview of the resulting screens. However, both suggestions are neither meant to be editable nor mandatory for the app creation process and rather serve as reassuring validation while modelling the use case. It can therefore be said that modelling activities should suit the users' previous experience, potentially ignoring established concepts of technical domains for the greater good of a more comprehensible and seamless modelling environment.

As a result, bringing mobile app modelling to this new level of abstraction not only bridges the gap to the field of business process modelling but can also impact organizations. On the one hand, new technical possibilities arise from process-centric app models. For example, already documented business processes can be used as input for cross-platform development targeting a variety of heterogeneous mobile devices. On the other hand, codeless app generation creates the opportunity for different development methodologies. The distinction between app developer and framework developer can lead to performance

benefits and better resource utilization on hardware-constrained devices such as smartphones. Best practices of mobile software development can be adopted by developers with expert knowledge of the respective platforms within the transformations which are then applied consistently throughout all generated apps. It has been shown that structural implementation decisions and even small-scale code refactorings can significantly improve battery consumption and execution times [52,53]. Also, instead of involving domain experts only in requirements phases before the actual development, an equitable relationship with fast development cycles is possible because changes to the model can be deployed instantly. Furthermore, future non-technical users may themselves develop applications according to their needs, extending the idea of self-service IT to its actual development. All of these ideas, however, rely on the modelling support provided by the environment, as begun with MAML's data model inference mechanism. Smart software to guide and validate the created models is required instead of simply representing the digital equivalent of a sheet of paper. In the future, graphical editors may evolve beyond just organizing and linking different models, towards tools enabling novel digital ecosystems through supportive technology.

7 Conclusion

In this work, a model-driven approach to mobile app development called MAML was presented which focuses around a declarative and platform-agnostic DSL to graphically create mobile business apps. The visual editor component provides advanced modelling support such as suggestions and validation through automatic data model inference. In addition, transformations allow for a codeless generation of app source code for multiple platforms. To evaluate the notation with regard to comprehensibility and usability, an extensive observational study with 26 participants was performed. The results confirm the design goals of achieving a wide-spread comprehensibility of MAML models for different audiences of software developers, process modellers, and domain experts. In comparison to the IFML notation, an equivalent MAML model is perceived as much less complex – in particular by non-technical users – and participants felt a high level of control, thus confidently solving their tasks. Furthermore, we analysed the challenges when extending the cross-platform approach to multiple app-enabled device classes. The applicability of MAML for this so-called *pluri-platform development* was assessed using a second study on a newly developed generator for the Wear OS smartwatch platform. As a result, MAML's approach of describing a mobile app as process-oriented set of use cases reaches a suitable balance between the technical intricacies of cross-platform app development and the simplicity of usage through the high level of abstraction and can be used to create app source code for both device classes from the same input models.

In case of the presented study results, some limitations may threaten their validity. Although a reasonable amount of participants was chosen for the observational interviews, additional evaluations may be carried out after the next iteration of MAML's development. Also, our participants were mostly students

which potentially reduces the generalizability of the results. However, their generation of app-experienced adults already participates in the general workforce and can be seen as realistic (albeit not representative) sample. The synthetic examples within the case study were designed to test a wide range of MAML's capabilities and uncover usability issues. Therefore, a real-world application would strengthen the validity of the approach and at the same time represents future work.

Regarding limitations of the approach itself, the chosen level of abstraction requires assumptions on the generic representation of data in the prototype. Possibilities to customize low-level details such as UI styling for different device classes need to be addressed in future, for example on the level of the intermediate MD^2 representation. Also, improvements of the generator prototype itself are part of ongoing work to provide a wide set of platform-adapted representations.

The presented process-oriented DSL offers the opportunity for research on a suitable framework structure for pluri-platform development and possible reuse of common transformations among multiple generators. Also, the process of developing such a framework of coupled components through a team with different roles may be investigated to further integrate model-driven techniques with traditional software development. Technically, further iterations on the framework's development are planned in order to provide additional user support, improve performance, and incorporate feedback based on the observed usability issues. Finally, the applicability of our approach to create business apps through model-driven transformations of MAML's platform-agnostic models to further device classes with drastically different UIs such as smart virtual assistants also presents exciting possibilities for future research.

References

1. van der Aalst, W.: Formalization and verification of event-driven process chains. Inf. Softw. Technol. **41**(10), 639–650 (1999). https://doi.org/10.1016/S0950-5849(99)00016-6
2. Alulema, D., Iribarne, L., Criado, J.: A DSL for the development of heterogeneous applications. In: FiCloudW, pp. 251–257 (2017)
3. Apache Software Foundation: Apache Cordova documentation (2019). https://cordova.apache.org/docs/en/latest/
4. Apple Inc: watchOS (2019). www.apple.com/watchos/
5. Architecture Board ORMSC: Model driven architecture (MDA): Document number ormsc/2001-07-01 (2001). http://www.omg.org/cgi-bin/doc?ormsc/2001-07-01
6. Bangor, A., Kortum, P., Miller, J.: Determining what individual SUS scores mean: adding an adjective rating scale. J Usability Stud. **4**(3), 114–123 (2009)
7. Bargh, J.A., Chartrand, T.L.: Studying the mind in the middle: a practical guide to priming and automaticity research. In: Judd, C.M., Reis, H.T. (eds.) Handbook of Research Methods in Social and Personality Psychology, pp. 253–285. Cambridge University Press, New York (2000)

8. Barnett, S., Avazpour, I., Vasa, R., Grundy, J.: A multi-view framework for generating mobile apps. In: IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), pp. 305–306 (2015). https://doi.org/10.1109/VLHCC.2015.7357239
9. Bizness Apps: Mobile app maker—bizness apps (2019). http://biznessapps.com/
10. Brambilla, M., Dosmi, M., Fraternali, P.: Model-driven engineering of service orchestrations. In: 5th World Congress on Services (2009). https://doi.org/10.1109/SERVICES-I.2009.94
11. Breu, R., Kuntzmann-Combelles, A., Felderer, M.: New perspectives on software quality [guest editors' introduction]. IEEE Softw. **31**(1), 32–38 (2014). https://doi.org/10.1109/MS.2014.9
12. Brooke, J.: SUS-a quick and dirty usability scale. In: Jordan, P.W., Thomas, B., Weerdmeester, B.A., McClelland, A.L. (eds.) Usability Evaluation in Industry, pp. 189–194. Taylor and Francis, London (1996)
13. Bubble Group: Bubble - visual programming (2019). https://www.bubble.is/
14. Buchmann, T.: Bxtend - a framework for (bidirectional) incremental model transformations. In: 6th International Conference on Model-Driven Engineering and Software Development (MODELSWARD) (2018). https://doi.org/10.5220/0006563503360345
15. da Silva, L.P., Brito e Abreu, F.: Model-driven GUI generation and navigation for android BIS apps. In: 2014 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD), pp. 400–407 (2014)
16. El-Kassas, W.S., Abdullah, B.A., Yousef, A.H., Wahba, A.M.: Taxonomy of cross-platform mobile applications development approaches. Ain Shams Eng. J. (2015). https://doi.org/10.1016/j.asej.2015.08.004
17. Esakia, A., Niu, S., McCrickard, D.S.: Augmenting undergraduate computer science education with programmable smartwatches. In: SIGCSE, pp. 66–71 (2015). https://doi.org/10.1145/2676723.2677285
18. France, R.B., Ghosh, S., Dinh-Trong, T., Solberg, A.: Model-driven development using UML 2.0: promises and pitfalls. Computer **39**(2), 59–66 (2006). https://doi.org/10.1109/MC.2006.65
19. Fuller, J.B., Hester, K., Barnett, T., Frey, L., Relyea, C., Beu, D.: Perceived external prestige and internal respect: new insights into the organizational identification process. Hum. Relat. **59**(6), 815–846 (2006). https://doi.org/10.1177/0018726706067148
20. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional Computing Series. Addison-Wesley, Reading (1995)
21. Gediga, G., Hamborg, K.C.: Evaluation in der software-ergonomie. J. Psychol. **210**(1), 40–57 (2002). https://doi.org/10.1026//0044-3409.210.1.40
22. GoodBarber: Goodbarber: Make an app (2019). https://www.goodbarber.com/
23. Google Inc: J2ObjC (2018). http://j2objc.org/
24. Google Inc: Wear OS by Google smartwatches (2019). https://wearos.google.com/
25. Granada, D., Vara, J.M., Brambilla, M., Bollati, V., Marcos, E.: Analysing the cognitive effectiveness of the WebML visual notation. Softw. Syst. Model. (2015). https://doi.org/10.1007/s10270-014-0447-8
26. Hemel, Z., Visser, E.: Declaratively programming the mobile web with Mobl. In: Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA), pp. 695–712. ACM (2011). https://doi.org/10.1145/2048066.2048121
27. International Organization for Standardization: ISO 5807:1985 (1985)

28. International Organization for Standardization: ISO 9241-110:2006 (2006)
29. Jesdabodi, C., Maalej, W.: Understanding usage states on mobile devices. In: ACM International Joint Conference on Pervasive and Ubiquitous Computing, UbiComp, pp. 1221–1225. ACM (2015). https://doi.org/10.1145/2750858.2805837
30. Jones, C., Jia, X.: The AXIOM model framework: transforming requirements to native code for cross-platform mobile applications. In: 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD). IEEE (2014)
31. Koren, I., Klamma, R.: The Direwolf inside you: end user development for heterogeneous web of things appliances. In: Bozzon, A., Cudre-Maroux, P., Pautasso, C. (eds.) ICWE 2016. LNCS, vol. 9671, pp. 484–491. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-38791-8_35
32. Langlois, B., Jitia, C.E., Jouenne, E.: DSL classification. In: The 7th OOPSLA Workshop on Domain-Specific Modeling (2007)
33. Ludei Inc: Canvas+ Cocoon documentation (2019). https://docs.cocoon.io/article/canvas-engine/
34. Majchrzak, T.A., Ernsting, J.: Reengineering an approach to model-driven development of business apps. In: Wrycza, S. (ed.) SIGSAND/PLAIS 2015. LNBIP, vol. 232, pp. 15–31. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24366-5_2
35. Majchrzak, T.A., Ernsting, J., Kuchen, H.: Achieving business practicability of model-driven cross-platform apps. OJIS **2**(2), 3–14 (2015)
36. Majchrzak, T.A., Wolf, S., Abbassi, P.: Comparing the capabilities of mobile platforms for business app development. In: Wrycza, S. (ed.) SIGSAND/PLAIS 2015. LNBIP, vol. 232, pp. 70–88. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24366-5_6
37. Mernik, M., Heering, J., Sloane, A.M.: When and how to develop domain-specific languages. ACM Comput. Surv. **37**(4), 316–344 (2005). https://doi.org/10.1145/1118890.1118892
38. Moody, D.: The "physics" of notations: towards a scientific basis for constructing visual notations in software engineering. IEEE Trans. Softw. Eng. **35**(5), 756–778 (2009)
39. Neate, T., Jones, M., Evans, M.: Cross-device media: a review of second screening and multi-device television. Pers. Ubiquitous Comput. **21**(2), 391–405 (2017). https://doi.org/10.1007/s00779-017-1016-2
40. Object Management Group: Business process model and notation (2011). http://www.omg.org/spec/BPMN/2.0
41. Object Management Group: Unified modeling language (2015). http://www.omg.org/spec/UML/2.5
42. Pentaho Corp: Data integration - kettle (2017). http://community.pentaho.com/projects/data-integration/
43. Product Hunt: 7 tools to help you build an app without writing code (2016). https://medium.com/product-hunt/7-tools-to-help-you-build-an-app-without-writing-code-cb4eb8cfe394
44. Reiter, A., Zefferer, T.: Power: a cloud-based mobile augmentation approach for web- and cross-platform applications. In: CloudNet, pp. 226–231. IEEE (2015). https://doi.org/10.1109/CloudNet.2015.7335313
45. Rieger, C.: Business apps with MAML: a model-driven approach to process-oriented mobile app development. In: Proceedings of the 32nd Annual ACM Symposium on Applied Computing, pp. 1599–1606 (2017)

46. Rieger, C.: Evaluating a graphical model-driven approach to codeless business app development. In: 51st Hawaii International Conference on System Sciences (HICSS), pp. 5725–5734 (2018)
47. Rieger, C.: MAML code respository (2019). https://github.com/wwu-pi/maml
48. Rieger, C., Kuchen, H.: A process-oriented modeling approach for graphical development of mobile business apps. Comput. Lang. Syst. Struct. **53**, 43–58 (2018). https://doi.org/10.1016/j.cl.2018.01.001
49. Rieger, C., Kuchen, H.: A model-driven cross-platform app development process for heterogeneous device classes. In: 52nd Hawaii International Conference on System Sciences (HICSS), pp. 7431–7440 (2019)
50. Rieger, C., Majchrzak, T.A.: A taxonomy for app-enabled devices: mastering the mobile device jungle. In: Majchrzak, T.A., Traverso, P., Krempels, K.-H., Monfort, V. (eds.) WEBIST 2017. LNBIP, vol. 322, pp. 202–220. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-93527-0_10
51. Rivera, J., van der Meulen, R.: Gartner says by 2018, more than 50 percent of users will use a tablet or smartphone first for all online activities (2014). http://www.gartner.com/newsroom/id/2939217
52. Rodriguez, A., Mateos, C., Zunino, A.: Improving scientific application execution on android mobile devices via code refactorings. Softw. Pract. Exp. **47**(5), 763–796 (2017). https://doi.org/10.1002/spe.2419
53. Sahar, H., Bangash, A.A., Beg, M.O.: Towards energy aware object-oriented development of android applications. Sustain. Comput. Inform. Syst. **21**, 28–46 (2019). https://doi.org/10.1016/j.suscom.2018.10.005
54. Simons, C., Wirtz, G.: Modeling context in mobile distributed systems with the UML. J. Vis. Lang. Comput. **18**(4), 420–439 (2007). https://doi.org/10.1016/j.jvlc.2007.07.001
55. Singh, K., Buford, J.: Developing WebRTC-based team apps with a cross-platform mobile framework. In: IEEE CCNC (2016). https://doi.org/10.1109/CCNC.2016.7444762
56. Spool, J., Schroeder, W.: Testing web sites: five users is nowhere near enough. In: CHI 2001 Extended Abstracts on Human Factors in Computing Systems, pp. 285–286. ACM (2001). https://doi.org/10.1145/634067.634236
57. The Eclipse Foundation: Sirius (2019). https://eclipse.org/sirius/
58. Thomson, G.: BYOD: enabling the chaos. Netw. Secur. **2012**(2), 5–8 (2012). https://doi.org/10.1016/S1353-4858(12)70013-2
59. Umuhoza, E., Brambilla, M.: Model driven development approaches for mobile applications: a survey. In: Younas, M., Awan, I., Kryvinska, N., Strauss, C., Thanh, D. (eds.) MobiWIS 2016. LNCS, vol. 9847, pp. 93–107. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-44215-0_8
60. Vaupel, S., Taentzer, G., Harries, J.P., Stroh, R., Gerlach, R., Guckert, M.: Model-driven development of mobile applications allowing role-driven variants. In: Dingel, J., Schulte, W., Ramos, I., Abrahão, S., Insfran, E. (eds.) MODELS 2014. LNCS, vol. 8767, pp. 1–17. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11653-2_1
61. Virzi, R.A.: Refining the test phase of usability evaluation: how many subjects is enough? Hum. Factors **34**(4), 457–468 (1992)
62. WebRatio: WebRatio (2019). http://www.webratio.com

63. Wolber, D.: App inventor and real-world motivation. In: 42nd ACM Technical Symposium on Computer Science Education (SIGCSE) (2011). https://doi.org/10.1145/1953163.1953329
64. Xamarin Inc: Developer center - Xamarin (2019). https://developer.xamarin.com
65. Zyła, K.: Perspectives of simplified graphical domain-specific languages as communication tools in developing mobile systems for reporting life-threatening situations. Stud. Log. Gramm. Rhetor. **43**(1) (2015). https://doi.org/10.1515/slgr-2015-0048