Proceedings of the 51st Hawaii International Conference on System Sciences | 2018

Evaluating a Graphical Model-Driven Approach to Codeless Business App Development

Christoph Rieger
ERCIS, University of Münster
Münster, Germany
Email: christoph.rieger@ercis.de

***Abstract*—Despite the growing interest in mobile app development, the creation of apps still follows traditional software development practices. Business apps are used by non-technical users in everyday work routines. However, their development is exclusively performed by software developers that need to centrally collect requirements and domain knowledge. Recent advances such as textual domain-specific languages (DSL) for cross-platform app generation reduce development efforts, but still focus on technical users. To alleviate these problems, the Münster App Modeling Language (MAML) is proposed as novel graphical DSL for specifying business apps. For each task to be accomplished within the app, the abstract process flows are modelled together with the respective data elements and view specifications in a combined model. Consequently, also non-technical users can express their domain knowledge without dealing with software engineering specifics. In contrast to existing process modelling notations, the MAML framework then allows for a codeless generation of apps for multiple platforms through model transformations and code generators. In order to automatically generate apps, the notation has to balance technical specificity and graphical simplicity. To assess the comprehensibility and usability of MAML's DSL, a qualitative usability evaluation was performed with software developers, process modellers, and domain experts.**

I. Introduction

Model-Driven Software Development has sparked significant interest in the past years. Through the use of models as primary software artefacts, increased efficiency and flexibility are only two of the expected benefits [1]. In the domain of mobile business apps, several approaches have been researched in academic literature such as MD^2 [2], Mobl [3], and AXIOM [4]. Those approaches provide cross-platform development functionalities using one common model for multiple target platforms, supporting current trends such as Bring your own device [5]. However, most of them focus on a textual Domain-Specific Language (DSL) to specify apps, often simplifying app development significantly but still restricting app creation to users with programming skills [6]. From a software engineering perspective, this predominantly top-down approach aligns well with traditional software development practices but may not be ideal in the context of small-scale mobile apps. Business apps focus on specific tasks to be accomplished by employees. A centralized definition of such processes may therefore deviate from the individual user's needs and then imposes additional burdens on the workforce instead 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 result and strengthen their identification with the company [7].

The research company Gartner predicts that more than half of all company-internal mobile apps will be built using codeless tools by 2018 [8]. To tap into the full potential, mobile app development can benefit from the incorporation of people from all levels of the organization. Development tools should therefore be understandable to both programmers and domain experts. Regarding business apps, graphical notations 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 templates (e.g., [9][10]).

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 proposes and evaluates the Münster App Modeling Language (MAML) framework. Rooted in the Eclipse ecosystem, the DSL grammar is defined as an Ecore metamodel, and technologies such as QVT Operational and Xtend are used for the creation of Android and iOS apps.

The contributions of this paper are twofold: First, a graphical DSL is presented that allows for the visual definition of business apps. The codeless app creation capabilities are demonstrated using the MAML editor with advanced modelling support and a fully automatic generation of native app source code through a two-step model transformation process. Second, a usability study is performed to demonstrate the potential and intricacies of an integrated app modelling approach for a wide audience of process modellers, domain experts, and programmers.

The structure of the paper follows these contributions. After presenting related work in Section II, the proposed framework is presented in Section III. Section IV discusses the setup and results of the usability evaluation. In Section V, the findings and implications of MAML are discussed before concluding with a summary and outlook in Section VI.

URI: http://hdl.handle.net/10125/50606
ISBN: 978-0-9981331-1-9

II. Related work

Different approaches to cross-platform mobile app development have been researched. In general, five approaches can be distinguished [11]. Concerning runtime approaches, *mobile webapps* are browser-run web pages optimized for mobile devices but without native elements, *hybrid approaches* such as Apache Cordova [12] 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 [13] 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. [14], 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 [11] – the graphical approach JUSE4Android [15] uses annotated UML diagrams to generate the appearance of and navigation within object graphs, and Vaupel et al. [16] presented an approach focusing around role-driven variants of apps using a visual model representation. Other approaches such as AXIOM [4] and Mobl [3] provide textual DSLs to define business logic, user interaction, and user interface in a common model. A more extensive overview of current model-driven frameworks is provided by Umuhoza and Brambilla [17]. However, current approaches mostly rely on a textual specification which limits the active participation of non-technical users without prior training [18], and graphical approaches are 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 [2]. This intermediate step is however automated and requires no intervention by the user (see Section III-D).

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. [19]. Process workflows can for example be modelled using BPMN [20], Event-Driven Process Chains [21], or flowcharts [22]. 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 [23]. However, a trade-off is necessary to balance the ease of use for modellers with the richness of technical details for creating functioning apps. Moody [24] 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 [26] and a BPMN extension to orchestrate web services [27]. 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 is domain-specific to mobile apps and covers both structural (data model, views) and behavioural (business logic, user interaction) aspects.

Lastly, visual programming languages have been created for several domains such as data integration [28] but few approaches focus specifically on mobile apps. RAPPT combines a graphical notation for specifying processes with a textual DSL [29], and AppInventor provides a language of graphical building blocks for programming apps [30]. 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 additional domain knowledge. Considering commercial frameworks, support for visual development of mobile apps varies significantly. In practice, most upcoming tools [31] focus only on individual components such as back-end systems or content management, or support particular development phases such as prototyping. Start-ups such as Bizness Apps [32] and Bubble Group [33] 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 [9]. The WebRatio Mobile Platform also supports codeless generation of mobile apps through a combination of IFML, other UML standards, and custom notations [34]. In contrast, this work focuses on a significantly more abstract and process-centric modelling level as presented in the next section.

III. 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 have sufficient information to transform them into fully functional mobile apps. Therefore, 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 Section III-D.

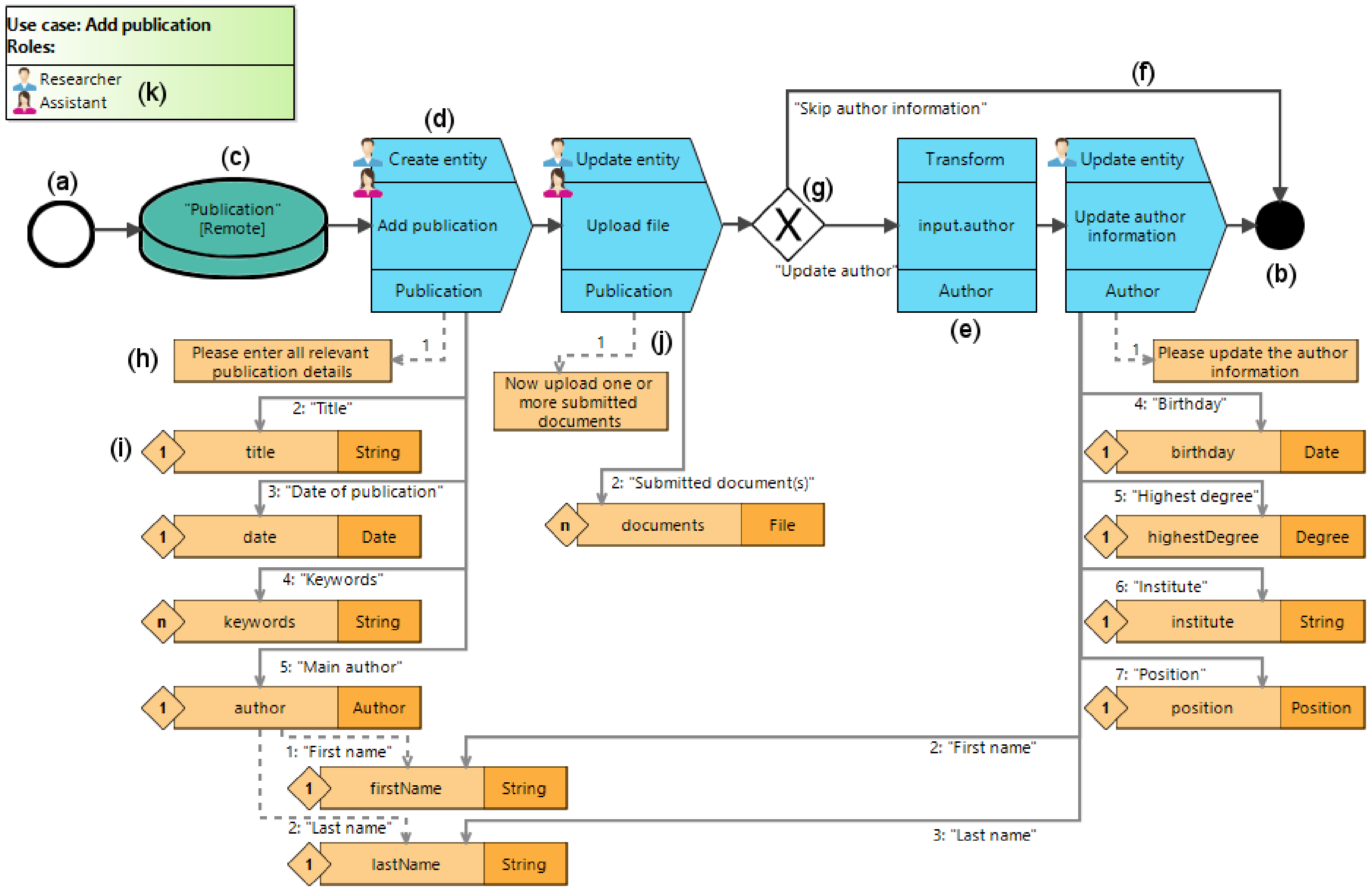


Fig. 1. MAML sample use case for adding a publication to a review management system [35]

A. 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 Section III-D) 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 [36]. 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 [37]. 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.

B. 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 Figure 1.

A model consists of a *start event* (labelled with (a) in Figure 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 Figure 1), or automatically by evaluating expressions referring to a property of the considered object.

The lower section of Figure 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 complex types, attributes may be nested over multiple levels (e.g., the "author" type in Figure 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 of 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 visualization of coherent 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 in their app.

C. 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 [38] that was extended to provide advanced modelling support for MAML.

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 [35] 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 Figure 1) is 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).

D. App generation

Technically, MAML is built using the Eclipse Modeling Framework (EMF), for example specifying the DSL's meta-model as an Ecore model. In order to generate apps, the proposed approach reuses previous work on MD^2 (see Section II). The complete generation process is depicted in Figure 2. First, a set of model transformations described in QVT Operational notation [39] is applied to transform graphical MAML models to the textual MD^2 representation. Amongst other activities, all relevant use cases are recombined, a complete 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.

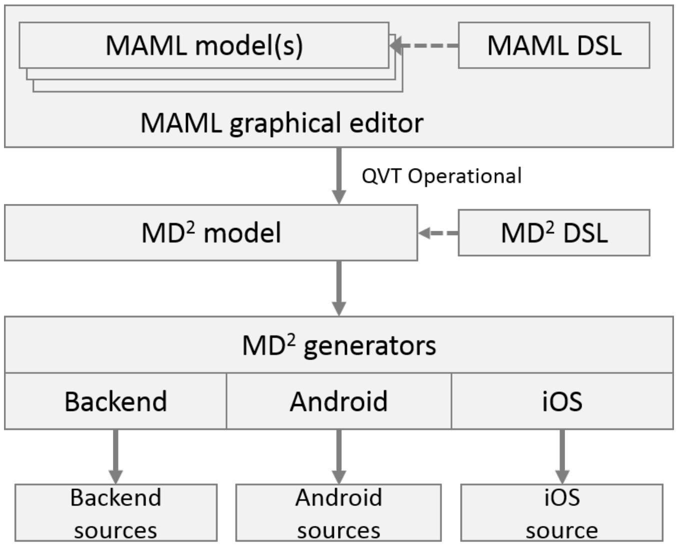
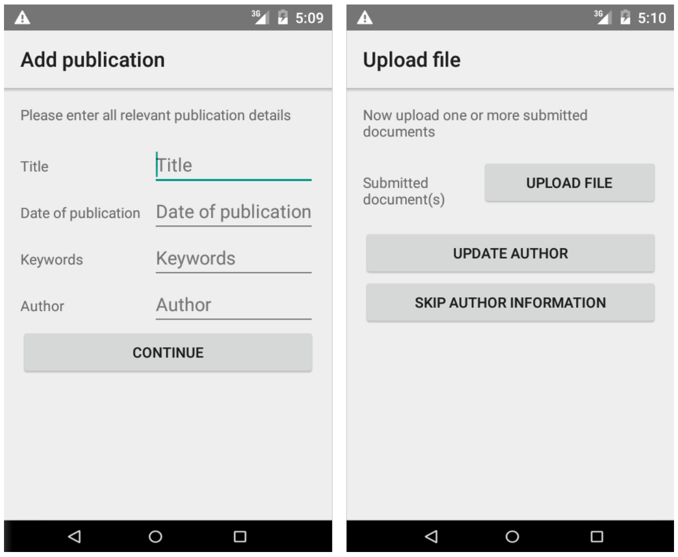


Fig. 2. MAML app generation process

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.

Do note that this proceeding differs from approaches such as UML's Model Driven Architecture [40] in that the intermediate representation is still a platform-independent representation but with a more technical focus that adds the possibility to modify default representations and configure parts of the application in more detail. 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. For example, the screenshots in Figure 3 depict the generated Android app views for the first process steps of the MAML model in Figure 1.



Please enter all relevant publication details

Title	Title
Date of publication	Date of publication
Keywords	Keywords
Author	Author

Now upload one or more submitted documents

Submitted document(s)

Fig. 3. Exemplary screenshots of generated Android app views.

IV. Evaluation

As demonstrated, MAML aligns with the goals (cf. Section III-A) of automated cross-platform app creation from modular and platform-agnostic app models. 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 Section IV-A, the results on comprehensibility and usability of the graphical DSL are presented.

A. 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 [41]. 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.

Therefore, 26 individual interviews of around 90 minutes 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 presented to calculate a System Usability Score (SUS) for each notation (cf. Section IV-B) [42]. 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 [43] (cf. Section IV-C).

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, effectivity, and efficiency. In total, over 1500 positive or negative observations were recorded as well as additional usability feedback and proposals for improvement.

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

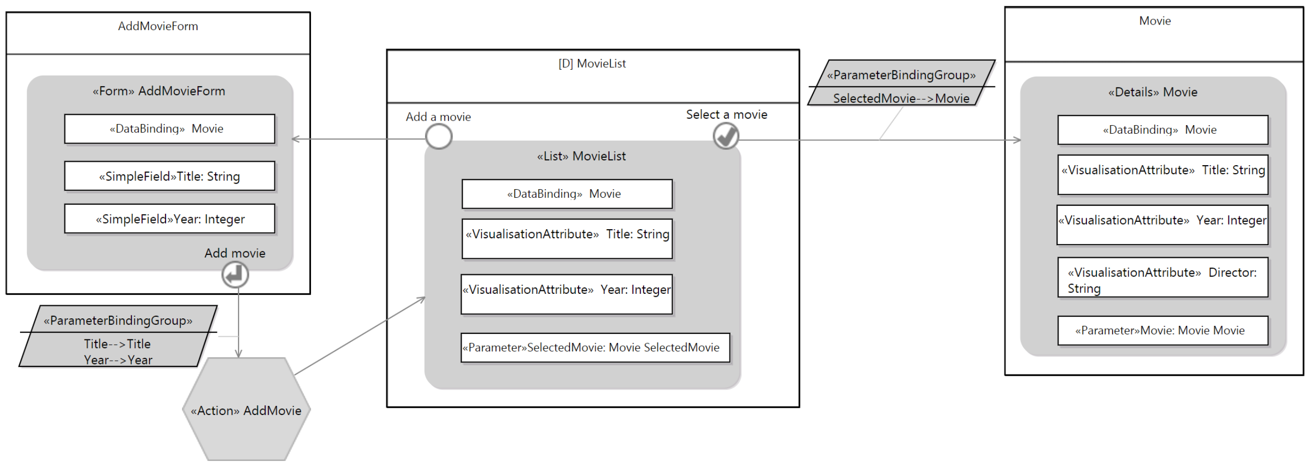

AddMovieForm
«Form» AddMovieForm
«DataBinding» Movie
«SimpleField»Title: String
«SimpleField»Year: Integer
Add movie
«ParameterBindingGroup»
Title-->Title
Year-->Year
«Action» AddMovie
[D] MovieList
Add a movie
Select a movie
«List» MovieList
«DataBinding» Movie
«VisualisationAttribute» Title: String
«VisualisationAttribute» Year: Integer
«Parameter»SelectedMovie: Movie SelectedMovie
«ParameterBindingGroup»
SelectedMovie-->Movie
Movie
«Details» Movie
«DataBinding» Movie
«VisualisationAttribute» Title: String
«VisualisationAttribute» Year: Integer
«VisualisationAttribute» Director: String
«Parameter»Movie: Movie Movie

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

traditional/web/app programming or data modelling, 9 process modellers have at least medium knowledge in process modelling (exceeding their development 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 [44] statement of five participants being sufficient to uncover 80% of usability problems in a particular software holds true [45], 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 hours per week, out of which 16.3 hours are spent on apps. In contrast, tablet use is rather low with 3.5 hours (3.2 hours for apps), and notebook usage is generally high with 27.5 hours but only 4.7 hours 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.

B. 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 need for extensive training. To compare the results with an existing modelling notation, equivalent IFML (see Figure 4) and MAML models [46] 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, excluding anchor effects and potential influences from a 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 [42]. The participants' scores for both languages and the respective standard deviations are depicted in Table I.

TABLE I
SYSTEM USABILITY SCORES FOR IFML AND MAML

SUS ratings	IFML	MAML
All participants	52.79 (σ = 23.0)	66.83 (σ = 15.6)
Software developers	45.91 (σ = 23.6)	64.09 (σ = 17.3)
Process modellers	64.17 (σ = 19.0)	69.44 (σ = 12.0)
Domain experts	48.33 (σ = 24.5)	67.92 (σ = 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. [47] to interpret the results as depicted in Figure 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.

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

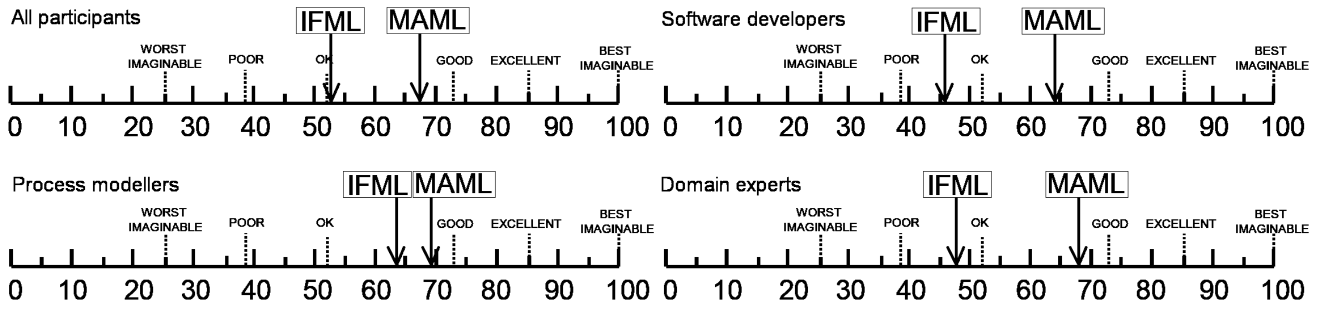


Fig. 5. SUS ratings for IFML and MAML.

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 device 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 process modelling languages 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 Section III-A, 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.

C. *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 most of MAML's features and concepts. In the hands-on context of a library app (cf. supplementary online material [46]), 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 following the ISO 9241-110 standard [43]. 35 questions with a scale between -3 and 3 cover the seven criteria of usability as presented in Table II. 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

TABLE II
ISONORM USABILITY QUESTIONNAIRE RESULTS FOR MAML.

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

describing this approach as uncommon or astonishing (see also Section V). Nevertheless, 67% of the participants state to have an understanding of the resulting app while modelling.

Self-descriptiveness refers to comprehension issues but additionally deals with the correct integration of different elements while modelling. For example, the concept of assigning roles was introduced to the participants but not the concrete usage. Still, 86% of them intuitively drag and drop role icons on process elements. 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, performing all activities in a single view instead of switching between multiple models. Many participants utter remarks such as "the editor does not evoke the impression of a complex tool". Parts of 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 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 performance or production-ready stability. 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 automatic 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 such details, 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 or read/modify relationship patterns.

V. 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. 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 [35]. 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. 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.

VI. 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, and 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 confirms 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. 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.

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. Our participants were mostly students, yet 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. 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, complex control flow logic and parallelism are so far not considered.

The presented process-oriented DSL offers the opportunity for research on transformations between MAML and process modelling notation such as BPMN in order to further integrate mobile app development with traditional business process management. 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, with the recent popularity of novel mobile devices such as smartwatches, their applicability to business apps through model-driven transformations of MAML's platform-agnostic models also present exciting possibilities for future research.

References

[1] T. Stahl and M. Völter, *Model-Driven Software Development*. Chichester: John Wiley & Sons, 2006.

[2] T. A. Majchrzak and J. Ernsting, "Reengineering an approach to model-driven development of business apps," in *8th SIGSAND/PLAIS EuroSymposium 2015, Gdansk, Poland*, 2015, pp. 15–31.

[3] Z. Hemel and E. Visser, "Declaratively programming the mobile web with Mobl," in *Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA)*. ACM, 2011, pp. 695–712.

[4] C. Jones and X. Jia, "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.

[5] G. Thomson, "BYOD: enabling the chaos," *Network Security*, vol. 2012, no. 2, pp. 5–8, 2012.

[6] M. Mernik, J. Heering, and A. M. Sloane, "When and how to develop domain-specific languages," *ACM Comput. Surv.*, vol. 37, no. 4, pp. 316–344, 2005.

[7] J. B. Fuller, K. Hester, T. Barnett, L. Frey, C. Relyea, and D. Beu, "Perceived external prestige and internal respect: New insights into the organizational identification process," *Human Relations*, vol. 59, no. 6, pp. 815–846, 2006.

[8] J. Rivera and R. van der Meulen, "Gartner says by 2018, more than 50 percent of users will use a tablet or smartphone first for all online activities," 2014. [Online]. Available: http://www.gartner.com/newsroom/id/2939217

[9] Xamarin Inc., "Developer center - Xamarin," 2017. [Online]. Available: https://developer.xamarin.com

[10] GoodBarber, "Goodbarber: Make an app," 2017. [Online]. Available: https://www.goodbarber.com/

[11] T. A. Majchrzak, J. Ernsting, and H. Kuchen, "Achieving business practicability of model-driven cross-platform apps," *OJIS*, vol. 2, no. 2, pp. 3–14, 2015.

[12] Apache Software Foundation, "Apache Cordova documentation," 2016. [Online]. Available: https://cordova.apache.org/docs/en/latest/

[13] Google Inc., "J2ObjC," 2016. [Online]. Available: http://j2objc.org/

[14] B. Langlois, C.-E. Jitia, and E. Jouenne, "DSL classification," in *The 7th OOPSLA Workshop on Domain-Specific Modeling*, 2007.

[15] L. P. da Silva and F. Brito e Abreu, "Model-driven GUI generation and navigation for android BIS apps," in *2014 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, 2014, pp. 400–407.

[16] S. Vaupel, G. Taentzer, J. P. Harries, R. Stroh, R. Gerlach, and M. Guckert, "Model-driven development of mobile applications allowing role-driven variants," *Lecture Notes in Computer Science*, vol. 8767, pp. 1–17, 2014.

[17] E. Umuhoza and M. Brambilla, "Model driven development approaches for mobile applications: A survey," in *Mobile Web and Intelligent Information Systems: 13th International Conference*, 2016, pp. 93–107.

[18] K. Żyła, "Perspectives of simplified graphical domain-specific languages as communication tools in developing mobile systems for reporting life-threatening situations," *Studies in Logic, Grammar and Rhetoric*, vol. 43, no. 1, 2015.

[19] R. Breu, A. Kuntzmann-Combelles, and M. Felderer, "New perspectives on software quality [guest editors' introduction]," *IEEE Software*, vol. 31, no. 1, pp. 32–38, 2014.

[20] Object Management Group, "Business process model and notation," 2011. [Online]. Available: http://www.omg.org/spec/BPMN/2.0

[21] W. van der Aalst, "Formalization and verification of event-driven process chains," *Information and Software Technology*, vol. 41, no. 10, pp. 639–650, 1999.

[22] International Organization for Standardization, "ISO 5807:1985," 1985.

[23] R. B. France, S. Ghosh, T. Dinh-Trong, and A. Solberg, "Model-driven development using UML 2.0: Promises and pitfalls," *Computer*, vol. 39, no. 2, pp. 59–66, 2006.

[24] D. Moody, "The "physics" of notations: Towards a scientific basis for constructing visual notations in software engineering," *IEEE Transactions on Software Engineering*, vol. 35, no. 5, pp. 756–778, 2009.

[25] D. Granada, J. M. Vara, M. Brambilla, V. Bollati, and E. Marcos, "Analysing the cognitive effectiveness of the WebML visual notation," *Software & Systems Modeling*, 2015.

[26] C. Simons and G. Wirtz, "Modeling context in mobile distributed systems with the UML," *Journal of Visual Languages and Computing*, vol. 18, no. 4, pp. 420–439, 2007.

[27] M. Brambilla, M. Dosmi, and P. Fraternali, "Model-driven engineering of service orchestrations," *5th World Congress on Services*, 2009.

[28] Pentaho Corp., "Data integration - kettle," 2017. [Online]. Available: http://community.pentaho.com/projects/data-integration/

[29] S. Barnett, I. Avazpour, R. Vasa, and J. Grundy, "A multi-view framework for generating mobile apps," *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pp. 305–306, 2015.

[30] D. Wolber, "App inventor and real-world motivation," *42nd ACM Technical Symposium on Computer Science Education (SIGCSE)*, 2011.

[31] Product Hunt, "7 tools to help you build an app without writing code," 2016. [Online]. Available: https://medium.com/product-hunt/7-tools-to-help-you-build-an-app-without-writing-code-cb4eb8cfe394

[32] Bizness Apps, "Mobile app maker — bizness apps," 2016. [Online]. Available: http://biznessapps.com/

[33] Bubble Group, "Bubble - visual programming," 2016. [Online]. Available: https://bubble.is/

[34] WebRatio, "WebRatio," 2017. [Online]. Available: http://www.webratio.com/site/content/en/home

[35] C. Rieger, "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*, 2017, pp. 1599–1606.

[36] Object Management Group, "Unified modeling language," 2015. [Online]. Available: http://www.omg.org/spec/UML/2.5

[37] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: Elements of reusable object-oriented software*, ser. Addison-Wesley professional computing series. Reading, Mass.: Addison-Wesley, 1995.

[38] The Eclipse Foundation, "Sirius," 2017. [Online]. Available: https://eclipse.org/sirius/

[39] ——, "QVT operational," 2017. [Online]. Available: http://www.eclipse.org/mmt/qvto

[40] Architecture Board ORMSC, "Model driven architecture (MDA): Document number ormsc/2001-07-01," 2001. [Online]. Available: http://www.omg.org/cgi-bin/doc?ormsc/2001-07-01

[41] G. Gediga and K.-C. Hamborg, "Evaluation in der software-ergonomie," *Journal of Psychology*, vol. 210, no. 1, pp. 40–57, 2002.

[42] J. Brooke, "SUS-a quick and dirty usability scale," *Usability evaluation in industry*, vol. 189, no. 194, pp. 4–7, 1996.

[43] International Organization for Standardization, "ISO 9241-110:2006."

[44] R. A. Virzi, "Refining the test phase of usability evaluation: How many subjects is enough?" *Hum. Factors*, vol. 34, no. 4, pp. 457–468, 1992.

[45] J. Spool and W. Schroeder, "Testing web sites: Five users is nowhere near enough," in *CHI '01 Extended Abstracts on Human Factors in Computing Systems*. ACM, 2001, pp. 285–286.

[46] C. Rieger, "Maml code respository," 2016. [Online]. Available: https://github.com/wwu-pi/maml

[47] A. Bangor, P. Kortum, and J. Miller, "Determining what individual sus scores mean: Adding an adjective rating scale," *J. Usability Studies*, vol. 4, no. 3, pp. 114–123, 2009.