Computer Languages, Systems & Structures 53 (2018) 43–58

Contents lists available at ScienceDirect

Computer Languages, Systems & Structures

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

A process-oriented modeling approach for graphical development of mobile business apps

Christoph Rieger*, Herbert Kuchen

ERCIS, University of Münster, Leonardo-Campus 3, Münster, 48149, Germany

ARTICLE INFO

Article history:
Received 12 August 2017
Revised 30 November 2017
Accepted 9 January 2018

Keywords:
Graphical DSL
Mobile application
Business app
Model-driven software development
Data model inference

ABSTRACT

Mobile app development is an activity predominantly performed by software developers. Domain experts and future users are merely considered in early development phases as source of requirements or consulted for evaluating the resulting product. In the domain of business apps, many cross-platform programming frameworks exist but approaches also targeted at non-technical users are rare. Existing graphical notations for describing apps either lack the simplicity to be understandable by domain experts or are not expressive enough to support automated processing. The MAML framework is proposed as model-driven approach for describing mobile apps in a platform-agnostic fashion not only for software developers but also for process modelers and domain experts. Data, views, business logic, and user interactions are jointly modeled from a process perspective using a graphical domain-specific language. To aggregate multiple use cases and provide advanced modeling support, an inference mechanism is utilized to deduce a global data model. Through model transformations, native apps are then automatically generated for multiple platforms without manual programming. Our approach is compared to the IFML notation in an observational study, with promising results regarding readability and usability.

© 2018 Elsevier Ltd. All rights reserved.

1. Introduction

A decade after Apple triggered the trend towards smartphones with its first iPhone, mobile devices and apps have been widely adopted. Business apps usually cover small-scale tasks and support the digitalization of processes which benefit from the increased mobility and availability of ubiquitous devices. For example, salespersons can access company data from a customer's location, expenses can be followed up remotely, and employees can submit requests for vacations not only from their office desk.

Until now, app development remains a task predominantly executed by programmers, often considering other stakeholders and future users primarily in requirements engineering phases upfront implementation. However, the research institute Gartner predicted that within a few years, more than half of all company-internal business apps will be created using codeless tools [1]. Many frameworks for *programming* mobile apps have emerged over the past years and cross-platform approaches allow for a large user base with low development efforts (an overview is given in [2–4]). Several commercial platforms provide cross-platform capabilities but usually focus on source code transformations, partly supported by graphical editors for designing individual views (e.g., [5,6]).

* Corresponding author.
E-mail address: christoph.rieger@ercis.de (C. Rieger).

https://doi.org/10.1016/j.cl.2018.01.001
1477-8424/© 2018 Elsevier Ltd. All rights reserved.

Modeling approaches that focus on platform-agnostic representations of mobile apps are rarely used in practice. Model-driven software development using a domain-specific language (DSL) bears the advantage of transforming a concise specification of the target application into a software product (semi-) automatically [7]. DSLs are generally suited to cover a well-defined scope with sensible abstractions for inherent domain concepts and increase productivity of developers compared to general purpose languages (GPL) [8]. Several textual DSLs for mobile apps have been presented in literature (e.g., [9–11]), although not all of them fully automate code generation [12]. However, *textual* DSLs provide only minor benefits to *non-technical users* because they still feel like programming [13].

Graphical modeling is therefore particularly suitable to describe apps by stakeholders with strong domain knowledge in order to better match the software product with their tacit requirements [14,15]. A suitable notation mandates a platform-independent design to also consider emerging mobile devices such as wearables, smart home applications, and in-vehicle apps.

To model sequences of activities, a wide variety of general-purpose process modeling notations such as Business Process Model and Notation (BPMN) exists [16]. Usually, those are not detailed enough to cover mobile-specific aspects and can hardly be interpreted by code generators from a technical point of view. In contrast, technical notations such as the Interaction Flow Modeling Language (IFML) are too complex to be understood by domain experts and require software engineering knowledge [17,18].

Moreover, the editor component is of major importance for the usability of a graphical modeling approach. Comparisons for graphical notations such as the Unified Modeling Language (UML) show that editors for the same notation differ significantly in modeling effort, learnability, and memory load for the user [19]. Editor development is a challenge in itself [20,21] and even the presence of a metamodel can only support the syntax of the resulting notation [22].

This article aims to alleviate the aforementioned problems by presenting the Münster App Modeling Language (MAML; pronounced 'mammal'), a *graphical DSL* for describing business apps that tackles the trade-off between technical complexity and graphical oversimplification in order to be understandable not only for software developers but also for domain experts and process modelers. *Model transformations* allow for a fully automatic generation of native smartphone apps for the Android and iOS platform from the specified graphical model without manually writing code. Besides the technical necessity of such a global model for code generation, it enables *advanced modeling support* for the graphical editor.

Four main research questions are addressed to investigate the potential of data model inference in the context of model-driven code generation approaches for mobile business apps:

(RQ1) How can user-oriented specification of business app functionality be achieved using a graphical modeling notation?
(RQ2) Is the modular subdivision of mobile app functionality feasible from a technical perspective with regard to the recombination of partial data models?
(RQ3) What additional support can data model inference provide for users to create semantically correct models?
(RQ4) Does the process-oriented subdivision of functionality help non-technical users in understanding and creating mobile app models?

Extending on previous work presented at SAC 2017 [23], this article presents the data model inference approach using a the scenario of an inventory management app and focuses on the benefits of such a mechanism for modeling environments regarding semantic semantic and validation (RQ3). In addition, the evaluation of MAML is extended to investigate the perceived usefulness of data model inference specifically for non-technical users (RQ4).

After presenting related work in Section 2, the proposed DSL and framework are presented in Section 3. Section 4 discusses the approach and presents evaluation results from a usability study before concluding in Section 5.

2. Related work

Since model-driven and cross-platform development of apps has been a topic for a few years now, there is plenty of scientific work on the general topic. However, only few graphical modeling approaches with subsequent code generation of mobile apps exist. Especially with regard to a workflow-related level of abstraction, related work on business process modeling is also considered in the following.

2.1. Cross-platform mobile apps

Developing mobile apps that run on multiple platforms can be achieved using different approaches. El-Kassas et al. [24] distinguish three major categories: *compiling* existing source code from a legacy application or different platform such as in [25], *interpreting* a single code base through a runtime or virtual machine such as Apache Cordova for developing hybrid apps [26], and *model-driven* generation of native app source code from a common representation. With regard to the latter category, various academic and commercial frameworks exist [12]. Only few of them, such as Mobl [9], PIMAR [27], and AXIOM [10] cover the full spectrum of runtime behavior and structure of an app; often providing a custom textual DSL for this means. The work in this article is based on MD^2 which focuses on the generation of business apps (i.e. form-based, data-driven apps interacting with back-end systems [4]). The input model is likewise specified using a platform-independent, textual DSL [11]. After preprocessing the model, code generators transform it into native platform source code for the Android and iOS platform [28,29]. Despite reducing development effort and complexity through domain-specific

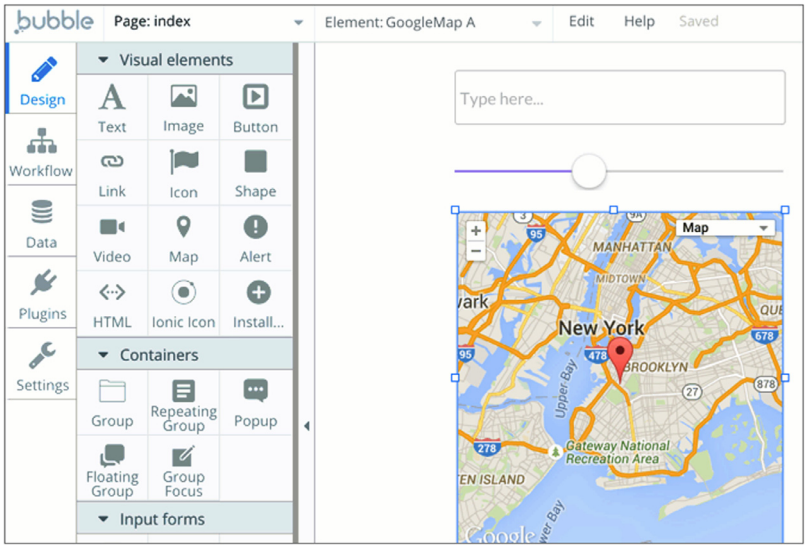

Fig. 1. Bubble's app configurator [32].

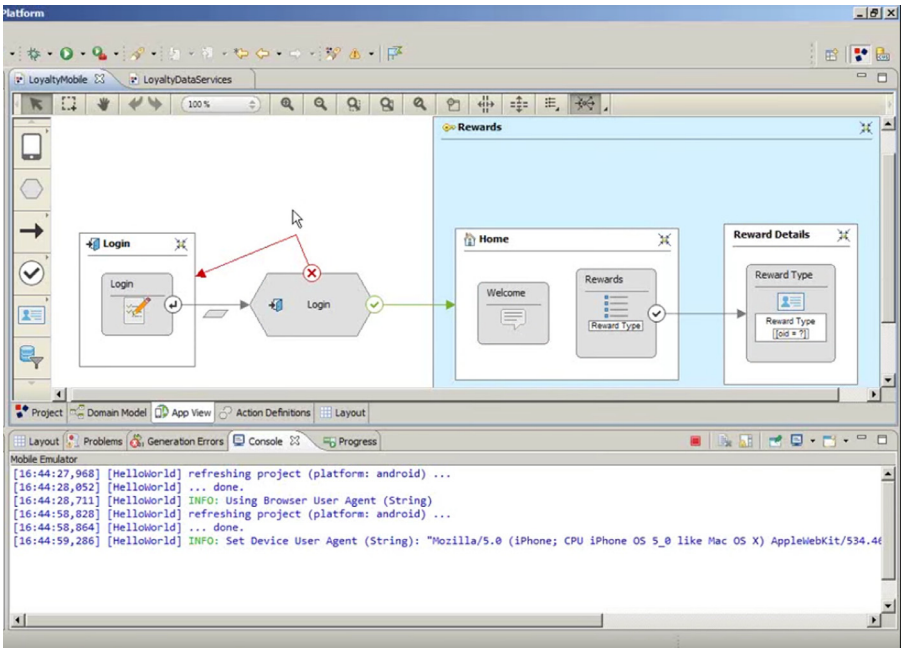

Fig. 2. WebRatio mobile platform [33].

concepts, textual DSLs often feel like programming to non-technical users [18]. In order to include those stakeholders and potential end users in the development of the app product, we consider graphical modeling of app contents [30,31].

Companies such as BiznessApps [34] and Bubble [32] (see Fig. 1) promise codeless creation of apps using detailed configurators and web-based user interface editors. Compared to these approaches, our work on the MAML framework abstracts from concrete interface specifications by focusing on a process-centric and platform-agnostic representation. The level of abstraction is therefore significantly higher such that modelers can focus on the logical flow of actions instead of positioning user interface (UI) elements on screen. The commercial WebRatio Mobile Platform [33] is closest to the work presented in this article by generating apps from a graphically edited model (see Fig. 2). It uses a multi-viewpoint combination of IFML, Unified Modeling Language (UML) class diagrams, and custom notations. This negatively influences learnability and comprehensibility for non-technical users who are first introduced to these technical notations (see Section 4). In contrast, we use an integrated modeling approach for specifying the sequence of process steps together with data objects and abstract UI information in a single model.

2.2. Process modeling notations

Graphical notations for supporting business operations have been developed for several purposes, including domains such as business process compliance [35] and data integration [36]. Specifically for mobile business applications, only few approaches exist. For example, AppInventor encourages novices to create apps by combining building blocks of a visual programming language [37], and Puzzle enables visual development of mobile applications within a mobile environment itself [38]. As semi-graphical approach, RAPPT represents a model-driven approach that mixes a graphical DSL for process flows with a textual DSL for programming business logic [39]. Although all of them aim at simplifying the actual programming work, they are often designed for novice or experienced developers and neglect non-technical stakeholders.

In contrast, general-purpose modeling notations exist to describe applications and process flows. The UML is the most prominent example for software developers, providing a set of standards to define the structure and runtime behavior of an application [40]. In particular, the IFML notation (succeeding WebML) represents user interactions within a software system [17]. To visualize process flows, a variety of modeling notations exist, e.g. BPMN [16], Event-driven Process Chains (EPC) [41], YAWL [42], or flowcharts [43]. As noted by Schalles [44], process modeling languages dealing with behavioral aspects are generally more usable than system modeling approaches that focus on structural concepts. However, process modeling notations often lack the technical specificity required for automated processing such as app creation. For example, the YAWL notation is constructed according to workflow management patterns [42] but initially did not provide a data perspective and does not support the specification of task types that could be used for app view templates.

Approaches specific to the Web or mobile domain, for example providing elements for mobile-specific functionality, rarely reach beyond UI modeling. Some related work explicitly tries to incorporate non-technical users, e.g. through a subdivision of models in content, form, navigation, and workflow viewpoints [45]. Franzago et al. [46] use a similar approach (separating data, UI, navigation, and business logic) for collaborative multi-viewpoint modeling. Also, a combination of multiple general-purpose notations can be used to describe all relevant application perspectives, for example in the model-driven approach by Koch et al. [47] based on multiple UML standards. Others use existing modeling notations such as statecharts [48] or extend them for mobile purposes, e.g. UML to model contextual information in mobile distributed systems [49], IFML with mobile-specific elements [50], or BPMN to orchestrate web services [51]. Yet, technical modeling notations are often considered as complex to understand for domain experts [52]. Cognitive studies have been conducted, e.g. for WebML [53], and problems such as symbol overload are aggravated by introducing multiple notations to represent the full range of data, business logic, UI, and user interaction modeling. As an example of integrating different application perspectives, Trætteberg and Krogstie [54] combine BPMN models with additional elements for representing the UI of desktop applications.

These works are mainly used to specify applications, not to automatically transform models in functional app source code. However, we deem process models to be a suitable level of abstraction for creating cross-platform apps with MAML.

2.3. Model aggregation

Further relevant fields of research for identifying commonalities in models include model differentiation techniques and schema matching of database structures [55,56]. According to the classification by Rahm [57], a schema-only and constraint-based approach on element level is suitable for the inference of a data model from multiple input models. For the given problem of partial models, we focus on an additive and name-based approach, although more sophisticated strategies such as ontology-based approaches may later improve the uncovering of modeling inconsistencies [58]. An application related to mobile devices, but focused on data visualization instead of its manipulation, is MobiMash for graphically creating mashup apps by configuring the representation and orchestration of data services [59].

In the context of metamodeling, reverse engineering approaches to track metamodel evolution have to cope with similar problems of inferring object structures [60] and finding correspondences from model instances [61]. López-Fernández et al. [62] presented a related idea of building a metamodel in a bottom-up fashion from sample model fragments.

3. MAML framework

The MAML framework consists of a graphical domain-specific language to specify mobile business apps as well as required tooling such as a model editor and code generators [63]. After presenting the overall design goals, these components and their interaction for seamless app generation are explained in the following.

3.1. Language design principles

The MAML DSL is built around five main principles:

Domain expert focus. Besides software developers, MAML should also be usable by non-technical users with knowledge about the circumstances of the target application, including for example process modelers and domain expert with limited software engineering experience. This requirement is relevant both for the notation itself and the surrounding tooling in order to engage with MAML without steep learning curve.

Data-driven process. MAML focuses on the domain of business apps. However, a process perspective is adopted by graphically describing an app as a sequence of processing steps performed on one or several data objects. This distinguishes MAML from similar approaches which provide far lower abstractions from the actual app development process.

Modularization. Traditional software engineering promotes a separation of concerns with patterns such as Model-View-Controller (MVC) [64]. To accommodate for the domain experts' mindset, MAML-based apps are specified with data structures, process steps, and screen visualization combined in a single model. In addition, mobile apps are typically designed to solve small and specific tasks. Therefore, processes are modeled as *Use Case*s, units of useful functionality with a self-contained set of behaviors and interactions performed by the app user [40].

Declarative description. Because of its platform-agnostic nature, MAML use cases describe *what* modifications should be performed on data objects. An imperative specification of *how* to perform the operation on a specific mobile device is not part of the model. The concrete representation depends on the target platform and suitable defaults are automatically provided during source code generation.

Automatic cross-platform app generation. An essential feature of the MAML framework is the automatic transformation of graphical models into fully-functional apps for multiple platforms. Cross-platform code generation constitutes an extensible approach to reach a large user base equipped with heterogeneous mobile devices. It is therefore mandatory that MAML models contain all technical details regarding app structure and runtime behavior in order to achieve a "zero coding" solution.

3.2. Language overview

For a fictitious inventory management system, the administration of item data and stock levels represents a typical business process. In MAML, this task is modeled in a *use case* as depicted in Fig. 3. The model contains a sequence of activities, from a *start event* (labeled with (1) in Fig. 3) towards one or several *end events* (2). In the beginning, a *data source* (3) specifies the data type of the manipulated objects and whether they are only saved locally on the device or managed by the remote backend system. Data can then be modified through a pre-defined set of (arrow-shaped) *interaction process elements* (4), for instance to *select/create/update/display/delete entities*, show *popup messages*, or access device functionalities such as the *camera* and starting a phone *call*. It should be noted that no device-specific assumptions on the appearance of a step can be made from this platform-agnostic description of models. The code generator instead provides representations and functionalities according to the platform capabilities, e.g., display a *select entity* step using a list of all available objects as well as functionality to filter or search for specific entries. Furthermore, *automated process elements* (5) represent invisible processing steps without user interaction, e.g., interacting with RESTful *web services, including* other models for process reuse, or navigating through the object graph (*transform*).

The navigation between connected process steps happens using an automatically created "Continue" button. Alternative captions can be specified along the *process connectors* (6). Many workflows are nonlinear, therefore process flows can be branched out using an *XOR* element (7). To decide which path is chosen, a condition can either be evaluated automatically based on specified expressions, or let the app user decide by providing buttons for each possible process path (such as in the example).

The rectangular elements represent the data objects which are displayed within a respective process step. *Attributes* (8) consist of a cardinality indicator, a name and the respective data type. Besides pre-defined data types such as *String, Integer, Float, PhoneNumber, Location* etc., enumerations and custom types can be defined. Consequently, attributes may be nested over multiple levels in order to further describe the content of such a custom data type. *Labels* (9) provide explanatory text to be displayed on screen, and *computed attributes* (10) are used to calculate derived values at runtime based on other attributes.

In MAML, only those attributes are modeled which will be used in a particular process step, for instance for including it in the graphical user interface or using its value in web service calls. Although this concept seems surprising to programmers expecting fully specified data models, it simplifies the integrated modeling of process flows and data. Attributes which are neither updated nor read do not contribute to the resulting app and are therefore ignored.[1]

Two types of connectors exist for attaching data to process steps: Dotted arrows represent a *reading relationship* (11) whereas solid arrows signify a *modifying relationship* (12) with regard to the target element. This refers not only to attributes displayed either as read-only text or editable input field. The interpretation also applies in a wider sense, for instance concerning web service calls in which the server "reads" an input parameter and "modifies" information in the app through its response.

Every connector which is connected to an interaction process element also specifies an order of appearance and a field description. For convenience, a human-readable field description is derived from the attribute name by default but can be

[1] Such attributes may, however, be required for manually written business logic after the code generation step. In this case, respective attributes can be likewise introduced in the generated source code.

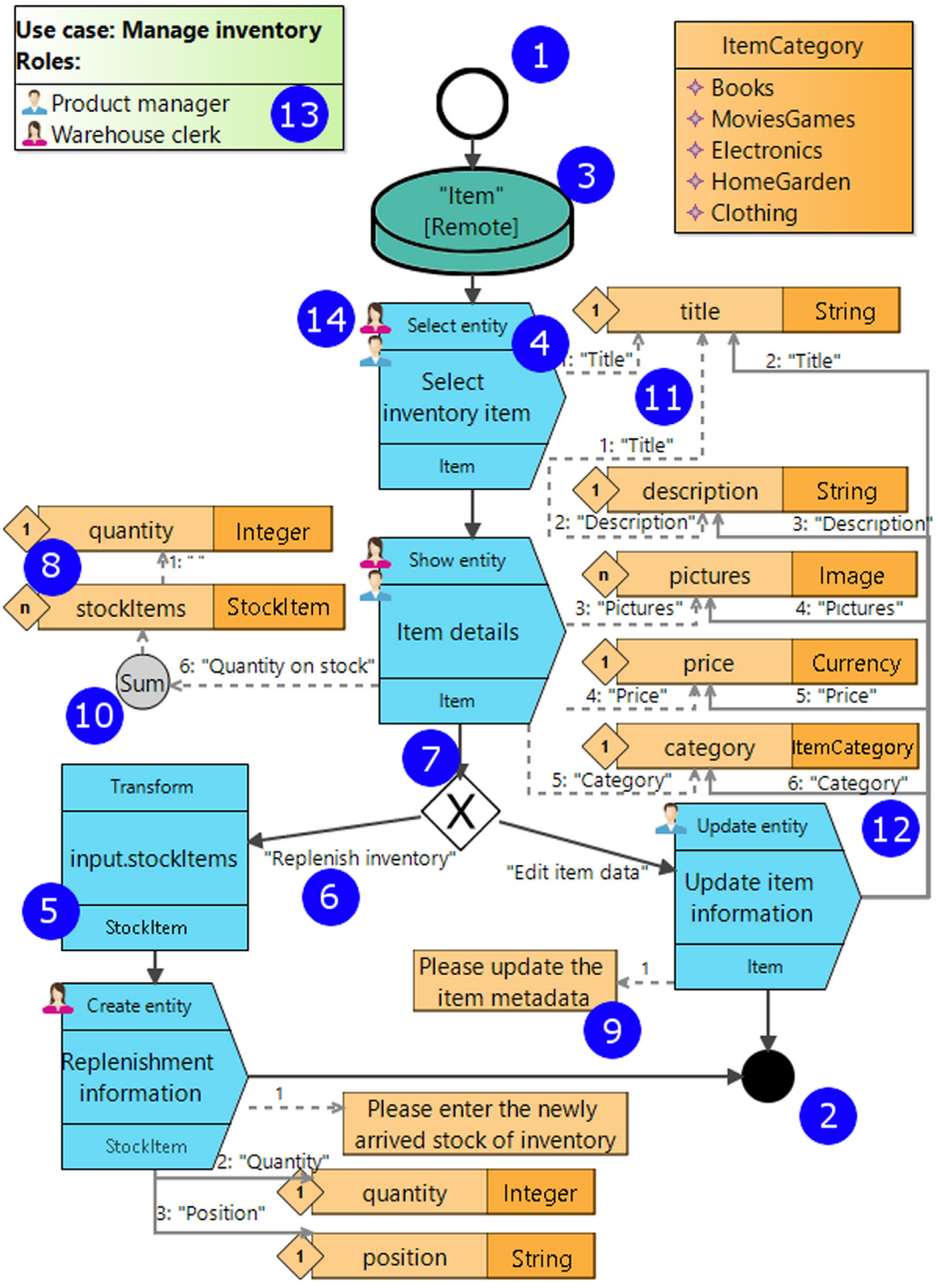

Fig. 3. MAML model with sample use case "Manage inventory".

modified. To reduce the amount of elements to model, multiple connectors may point to the same UI element from different sources (given their data types match). Alternatively, to avoid confusing connections across larger models, UI elements may instead be duplicated to different positions in the model and will automatically be matched in the inference process (see Section 3.3).

Finally, MAML supports a multi-role concept because many business processes such as approval workflows involve different people or departments. The modeler can specify arbitrary role names (13) and annotate them to the respective interaction process elements (14). If the assigned role changes, the generated app automatically terminates the process for the first app user, transfers modified data objects, and informs the subsequent user about an open workflow instance in his app.

3.3. Data-model inference

As described in the previous paragraphs, each process step within a model refers only to the attributes needed within this particular step. Besides avoiding redundancies of separately specified data models, this proceeding suits the mindset of non-technical users who focus on the task's process and consider separate data models "technical clutter". Also, due to the modular development, modelers might not even be aware of particular data fields required in other processes. Nevertheless, a global data model is a technical requirement for performing code generation. Therefore, the global view on all data structures in the resulting app is inferred from partial models on multiple levels: for each process element individually, then for the whole use case, and finally across multiple use cases.[2]

3.3.1. Partial data-model inference

In MAML, each process step effectively represents a partial data model describing the main data type of the step with the connected attributes. The data model inference approach therefore constructs a directed graph structure with vertices representing data types and edges denoting attribute relationships between two types.[3] For clarification, the term *attribute* is further used to refer to the UI element in a MAML model whereas *property* signifies the corresponding member of the UML class diagram representation.

Firstly, all data types of the process element and its attached attributes are collected. Fig. 4 depicts a sample process element with the set of (indirectly) related data types *Item, Company, ItemCategory, StockItem, String*, and *Integer*. For each non-primitive data type in this set, a class is created in the partial data model (including enumerations such as *ItemCategory*). Secondly, relationships between those data types are identified. Three origins need to be considered (cf. numbered circles in Fig. 4):

(1) A relationship may exist between a process element and an attached attribute.
(2) A nested attribute adds a relationship to the nesting attribute's data type.
(3) An attribute may be transitively connected to the process element through computed attributes, but still refers to the process element's data type.

The identified relationships are transformed into properties of the containing class by distinguishing the following four cases.

[2] However, semantic reasoning such as inferring generalization relationships and fuzzy matching techniques for identifying "similar" attributes are beyond the scope of this work.

[3] It should be noted that the implemented algorithm [65] is more complex than sketched here for conveying the idea in a concise way.

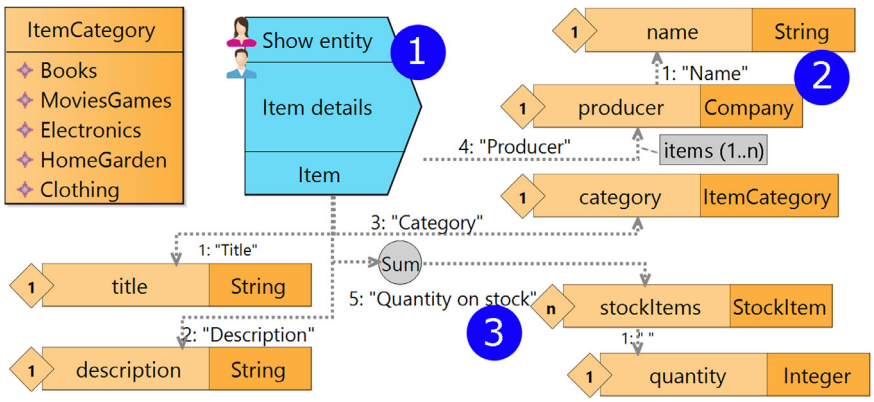

Fig. 4. Exemplary partial MAML model.

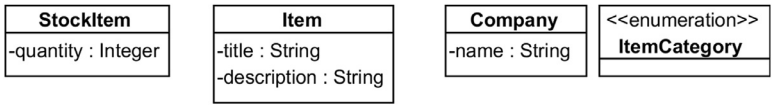

Fig. 5. Class diagram of inferred primitive types.

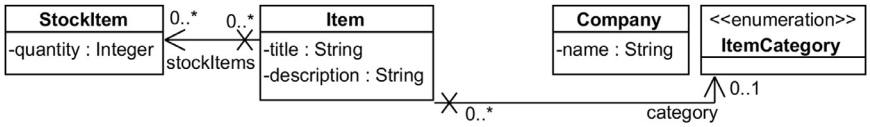

Fig. 6. Class diagram including inferred unidirectional relationships.

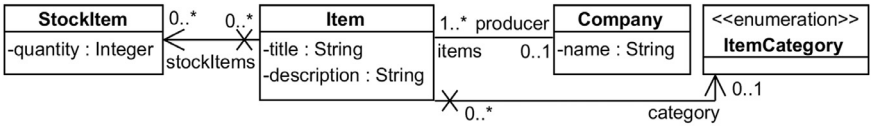

Fig. 7. Class diagram including inferred bidirectional relationships.

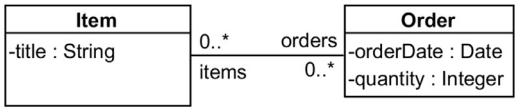

Fig. 8. Class diagram of a second MAML model.

Primitive. An attribute with primitive data type is converted to a single- or multi-valued property of the source data type. For the exemplary process step depicted above, *description, name, quantity* and *title*, are added as properties to the respective classes (see Fig. 5).

Unidirectional. Attribute connections are usually represented as unidirectional relationships. Because multiple connections may be present in different parts of the MAML model, the opposite direction is unknown and must be interpreted with unrestricted $0..*$ cardinality, where $i..j = \{i, i+1, ..., j\}$ and $*$ represents infinity. For example, the *stockItems* property in Fig. 6 holds a collection of *StockItem* objects but may itself be referenced by multiple *StockItem*s in other parts of the application.

Bidirectional. In contrast to the previous case, explicit bidirectional relationships in the model are transformed to named properties in both classes. For instance, the *producer* attribute in Fig. 4 represents a multi-valued property and is converted to the association shown in Fig. 7. In addition to creating access methods for consistent updates and deletions, potential cardinality restrictions need to be enforced in the generated code (e.g., forbid the removal of the last related *item* to comply with the minimum cardinality in the *company* object).

Singleton. An attribute of a singleton data type (not depicted in the example) is a variant of the unidirectional scenario in which the opposite cardinality is known to be restricted to 0..1 (i.e., either the property is set in the single instance of the referencing class or not) [64].

3.3.2. Merging partial data models

After creating the graph data structure for an individual process element, the multitude of partial data models needs to be merged both for the whole process within a use case and the overall app product consisting of multiple use cases. Because of the additive merging process, partial models can be merged iteratively in arbitrary order.

To continue the example of the previous section, the partial data model depicted in Fig. 7 is merged with another partial model depicted in Fig. 8. In a first step, the union of both sets of data types is created such that the global data model contains the types *Item, Company, ItemCategory, StockItem, Order, String*, and *Integer*.

Next, each edge of both partial models is added to the global data model if it introduces a new name or cardinality to the respective combination of source and target data type. With this name-based matching strategy, specifying multiple associations between the same pair of data types is unproblematic and will result in the same edge of the type graph. MAML modelers are therefore free to either link to the same attribute from different origins in the model or clone the modeled attribute for better readability. For the exemplary models, this results in the complete global data model depicted in Fig. 9.

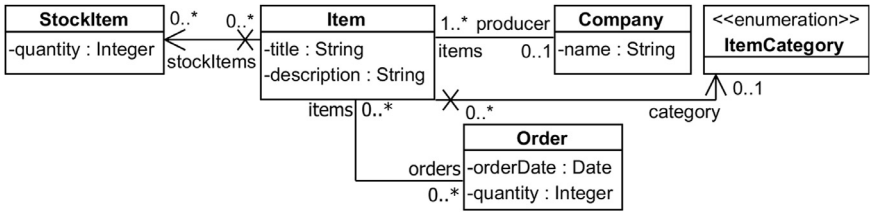

Fig. 9. Inferred global data model.

However, two types of modeling errors may occur when merging these data structures as visualized in Fig. 10:

(1) A *type error* exists if any source data type has two properties of the same name pointing to different target data types. Automatically resolving this error is impossible and the modeler needs to be informed.
(2) A *cardinality conflict* exists if two properties with the same name differ with regard to their cardinality for any pair of data types. In this case, the modeler should be warned, but automatic resolution is possible.

In case of cardinality conflicts, the resulting cardinality for each side of the association is calculated as the union between the conflicting cardinalities (ignoring not navigable ends) to avoid invalidating existing data. Regarding the inferred directionality, the merged association is bidirectional if any of the affected relationships is modeled to be bidirectional. In the example class diagram depicted in Fig. 11, the cardinality $0..1 \cup 0..* = 0..*$ is assigned to b and $1..*$ is assigned to a (as $0..*$ is not navigable in this direction).

To sum up, this inference algorithm can be applied to partial models of different granularity in order to derive a global data schema and simultaneously validate the individual models.

3.4. Modeling support

The developed editor for MAML is based on Eclipse Sirius and the underlying Eclipse Modeling Framework [66]. Beyond the structural constrains imposed by the DSL's metamodel, semantic restrictions need to be considered for creating valid models [67]. Language workbenches for developing DSLs usually provide additional features to help end users model "correct" models [68]. For example, in the Xtext framework for developing textual DSLs, *validators* can be defined to programmatically apply rules to model elements and *scope providers* allow for context-sensitive filtering of references [69]. Eclipse Sirius, as generic tool for graphical editor development, could be called an "editor workbench" and provides similar mechanisms for controlling the modeled result. These features can be classified into *validation* of the model itself and *semantic services* of the editor [68].

3.4.1. Supporting model validation

Different systems for categorizing the severity of faults are commonly used in software development, usually comprising a fatal error state, one or more levels of error severity regarding the implications on functionality, and one level of trivial issues [70,71]. Using the aforementioned data model inference approach, the developed tool support for MAML includes rules on corresponding levels of severity.

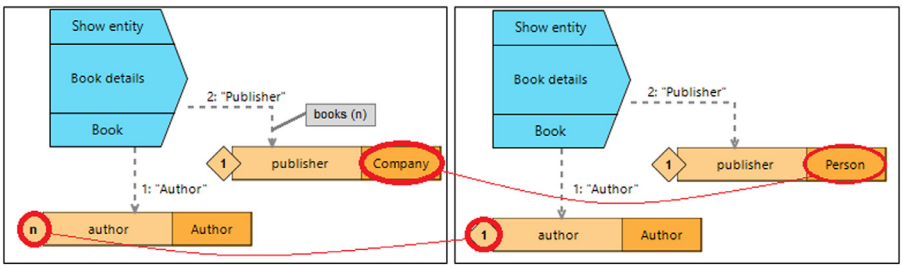

Fig. 10. Visualization of modeling errors [65].

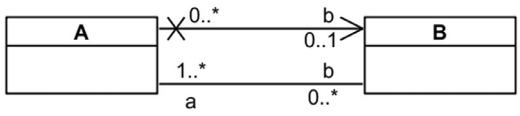

Fig. 11. Exemplary model with cardinality error.

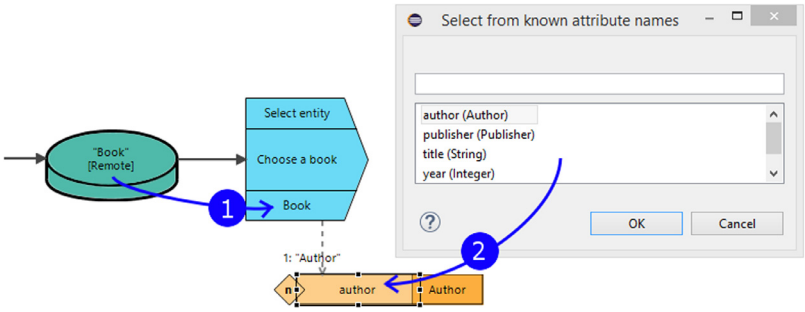

Fig. 12. Examples for semantic editing services.

Error. Blockers with regard to model editors signify the inability to execute or process the model. In addition to the graph structure which describes the partial data models of Section 3.3, the respective *attribute* in the MAML model is also tracked. Unresolvable type errors identified by the merge algorithm (cf. Section 3.3.2) can therefore be traced back to all affected model elements. This allows for a visual feedback of semantic discrepancies directly within the model.

Warning. Without further differentiating severity levels, warnings can be regarded as subsuming all model inaccuracies that are not technically incorrect but influence the processing result in probably unintended ways. For example, the self-resolved cardinality conflict in Section 3.3.2 might have unintended consequences in views that initially expected single-valued elements. Further, omissions such as defining but not assigning roles are considered as unintentional error. Obviously, the modeler also needs to be notified about warnings, ideally presenting possible alternatives to repair the modeling mistake.

Hint. Trivial issues do not affect the model outcome but are likewise displayed next to the respective model element in order to inform the user about potentially unwanted effects or consistency issues. In MAML, naming conventions, e.g., lowercasing attribute names, can seamlessly be integrated as hints in the editor.

3.4.2. Providing semantic services

In addition to validating models, global data model inference allows for semantic support within and across MAML models.

Continuous semantic assessment. The best level of modeling support is preventing the appearance of erroneous conditions in the first place. Invisible to the user, semantic discrepancies are checked when modifying or adding elements in the model. Because of the inferred knowledge, introducing semantic errors such as attaching incorrect data types is denied – similar to scope providers that provide a context-dependent set of valid elements. Also, textual expressions for specifying branching conditions of *XOR* elements or navigating through the data structure in a *Transform* element (cf. Fig. 3) are evaluated at runtime. Beyond the syntactic correctness, the correspondence to elements in the data model is immediately checked.

This real-time validation capability is also important for another type of continuous modeling support built into MAML: At the bottom of each process element, the data type of the manipulated objects is automatically inserted as mental assistance and additional guidance for non-experienced modelers (labeled (1) in Fig. 12). Whenever process flow elements are (re-)connected, the validity of the process chain is checked. For example, merging multiple branches that operate on different data types is automatically disallowed.

Context-sensitive suggestions. Finally, the inference results are used to reduce modeling effort similar to intelligent code completion features known from textual editors. For instance, matching attributes are suggested for process elements in case the source data type is known from other steps in the model or different use cases within the same app (see (2) in Fig. 12). Similarly, matching data types and cardinalities can automatically be completed for existing attributes.

To sum up, using the data inference mechanism not only prepares the MAML model for automatic code generation as described in the following section. It also strengthens the model's consistency while at the same time reducing the overall modeling effort. The graphical modeling environment thus evolves beyond the stage of just providing the infrastructure for modeling "shapes filled with text" into a context-aware tool that promptly reevaluates the semantic validity while models are built or modified.

3.5. App generation

One of the distinguishing features of the MAML framework is its automatic generation of apps for multiple target platforms in order to eliminate the need for manual programming. MAML relies on a model-driven software development approach and the graphical models are used as sole input for the app generation.

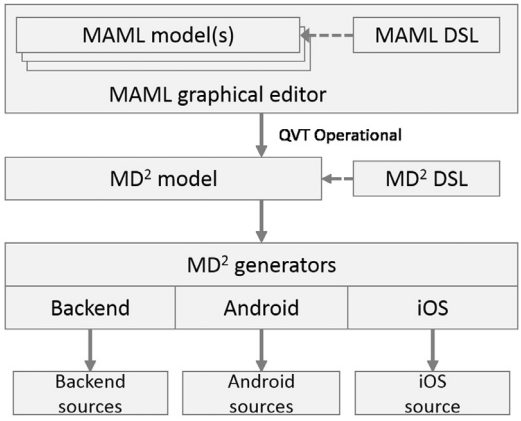

Fig. 13. MAML generation process [23].

The current approach is based on previous work in the domain of business apps, notably the textual DSL MD^2 (see Section 2). Although both languages share common overall concepts, MD^2 is programming-oriented and follows a MVC-layered approach instead of the presented separation by functionality. For further details on MD^2 language features the reader is referred to [11,28].

A two-step generation process was established (cf. Fig. 13): First, a model-to-model transformation translates all related MAML model instances to one MD^2 model instance. This transformation is based on mappings between the metamodels of MAML and MD^2, specified using the QVT Operational framework [72]. Because both DSLs are designed for the same domain of business apps, many concepts such as roles and process flows are present in both notations and can be transformed unambiguously. In addition, there are MD^2 concepts such as content providers or view layouts which have no correspondence because MAML operates on a higher level of abstraction. If mandatory for MD^2 models, default representations are created for these concepts during transformation.

Second, existing MD^2 generators perform the model-to-code transformation and output the app source code for the target platforms, in particular Java code for Android and Swift code for iOS. Fig. 14 presents exemplary screenshots of the resulting Android app for the first process steps shown in Fig. 3.

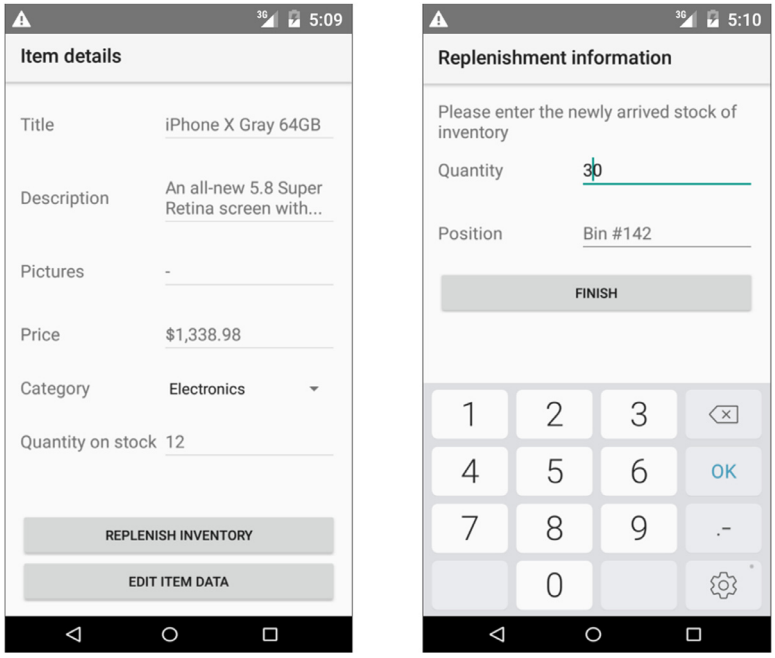

Fig. 14. Generated Android app screenshots.

4. Evaluation and discussion

With regard to RQ1 and RQ2, the design of MAML shows the possibility of replacing the software engineering focused separation of concerns in mobile app models by a business perspective using functional subdivision. The MAML framework also aims for zero-code generation of cross-platform apps from the graphical model, thus requiring a minimum of technical specificity to be interpretable by the data model inference process and subsequent generators. In this context, business processes are seen as reasonable level of abstraction. It is, however, not intended to create yet another workflow management system with the presented framework. Currently, variability in the process flow is deliberately limited to XOR elements and deemed sufficient due to the sequential display on smartphone screens.

There are other process modeling languages such as EPC with a high degree of usability [44] but which rely on humans to interpret the meaning of models. Yet, a rigid representation of technicalities such as information propagation through parameter declarations might alienate potential users with limited experience in software development. Finding a suitable trade-off between technical detailedness and visual simplicity [44] was therefore essential for designing MAML's modeling notation. This conflict can be alleviated by embedding domain knowledge and aligning with existing standard notations [7]. With regard to modeling complexity, the MAML DSL is positioned between BPMN 2.0 and IFML.

However, a platform-agnostic notation which is at the same time designed for mobile apps is not a contradiction. Conceptual differences exist both with regard to the characteristics of mobile platforms in general as well as the usage context. Regarding mobile specifics, the platforms are characterized by a heterogeneity of input and output mechanisms even among devices of the same platform – much more than desktop- or browser-based applications which rely on established mouse and keyboard inputs. For instance, smartphones typically provide capacitive displays but at the same time support different modes of touch input (e.g., multi-touch gestures or pressure-sensitive "3D touch" on Apple devices[4]) and can be controlled using additional hardware buttons or voice commands [73]. Device capabilities differ with regard to sensors such as gyroscopes for motion detection, determination of position via GPS, or front/back cameras. Also, seamless interaction with operating system functionality such as starting phone calls is possible [74]. Moreover, the usage behavior of mobile apps is reflected in the design of the DSL. For example, splitting apps by use cases and modeling workflow and data perspectives in a combined model is feasible for the scope of mobile processes but becomes unpractical for enterprise workflows such as production processes. The suitability of partial data models might be questioned in large-scale models, but with regard to app development, fully specified data models are often not pre-existing and need to be newly created. Although the notation can of course be used in non-mobile development scenarios which make use of MAML's independence from concrete UI representations, these mobile specifics were considered during its development in order to create an understandable notation primarily for describing mobile apps.

Compared to other modeling notations, MAML models have the advantage of being self-contained and do not rely on a synchronization with external information. In contrast, IFML models are connected to other UML standards and require multiple models for data, business logic, and user interaction to be interpreted together. In addition, all pieces of information are presented in an integrated model. MAML models can therefore also be seen as a means of communication, facilitating the discussion between involved stakeholders concerning the structure, interactions, and data of an app product. We do not claim that this design is better under all circumstances. However, we argue that the task-oriented approach chosen for MAML is reasonable for the purpose of small-scale processes performed in apps.

With regard to a comparable scope, IFML is the most similar approach to ours and can be compared to gain insights regarding RQ4. Although BPMN and further simplified notations [75] seem more usable at first sight, they lack significant capabilities to represent all perspectives of app development. Also, a notation with more elements does not necessarily lead to a degradation of its usability [44]. To compare IFML with MAML, a qualitative, observational study was performed with 26 (mostly student) participants in individual sessions of 90 min duration. None of the participants had previous knowledge of either modeling language.

The first part of the study evaluated the readability and understandability of each notation. Therefore, one MAML and one IFML model, both depicting similar scenarios (cf. online material [63]), were shown to the participants (in random order to avoid bias) without prior introduction to the notation. Additionally, a System Usability Scale (SUS) questionnaire was answered for each notation, comprising ten questions on a 5-point scale between strong disagreement and strong agreement. The result is obtained by converting and scaling the responses according to the method presented by Brooke [76].

Overall, MAML surpasses IFML with 66.8 to 52.8 points (of maximum 100) regarding its readability without prior knowledge. It should be noted that this score does not represent a percentage value but can instead be interpreted according to the adjective rating scale depicted in Fig. 15 as proposed by Bangor et al. [77].

To gain additional insights, the participants were clustered in the three groups of software developers (11), process modelers (9), and domain experts (6), according to their personal background in programming and process modeling. Although differences can be observed, all groups rated MAML better, as summarized by Table 1.

In more detail, aspects regarding the ease of understanding and complexity of the notation scored better for MAML, particularly for the group of domain experts without knowledge of programming or process models. This reflects the

[4] https://developer.apple.com/ios/3d-touch/.

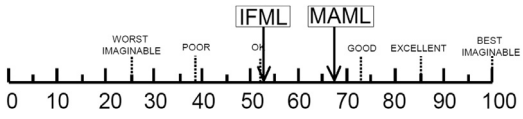

Fig. 15. SUS ratings for IFML and MAML [23].

Table 1
SUS scores by participant group.

Group	IFML score (σ)	MAML score (σ)
Software developers	45.91 (23.6)	64.09 (17.3)
Process modelers	64.17 (19.0)	69.44 (12.0)
Domain experts	48.33 (24.5)	67.92 (18.7)
Overall	52.79 (23.0)	66.83 (15.6)

observations in which seven participants emphasized MAML's simplicity and reduction of "technical clutter". Interestingly, even software developers generally preferred MAML's process-oriented approach. Only two participants critically noted the absence of explicit data models, supporting the idea of integrated process modeling. The comparatively low baseline for the group of software developers might result from a general disinclination to specify models for software products due to the complexity of technical notations, supplementary effort to the "actual" development, or experienced problems of keeping both artifacts synchronized over time [78]. Because of its model-driven approach that considers source code as derived artifact, these issues do not apply to MAML.

Data model inference as demonstrated in MAML alleviates three problems of graphical modeling: First, the modeler is disburdened of separately maintaining a redundant data model. On-the-fly inference of the underlying data model is particularly helpful for process modelers and domain experts without programming experience to quickly get started with modeling the actual app behavior. The study confirms this, as the inference mechanism and enhanced modeling support was welcomed as helpful guidance during the sessions, and scores favor MAML regarding consistency and explanatory power (RQ4).

Second, the availability of fine-grained partial data models (per process flow element) allows for improved security. Generators may tailor app-specific data models by splitting the global data model according to the required fields. This not only reduces the volume of transferred data but allows for automatically generated validation mechanisms to control data access permissions.

Third, improved modeling support as described in Section 3.4 is only possible if interrelations between elements are accessible to the editor component (RQ3). Many graphical editors focus on the usability of placing and connecting elements on the canvas but lack the ability to provide context-sensitive modeling support similar to integrated development environments for textual programming languages. Further sophisticated inference techniques relying on ontology-based matching [79] may be extensions to further improve model consistency.

When comparing the answers of domain experts and technical users (developers and process modelers together) for the SUS questionnaire (see Fig. 16; answers rescaled to an [0;4] interval), the contrasting approaches of using a technical (IFML) or a process-oriented (MAML) separation of models become apparent. Of course, a net effect of modeling support through the inference mechanism cannot be measured as it is intertwined with the graphical syntax of the notation itself (e.g., inferred type hints for process elements). However, responses are significantly higher for domain experts assessing whether the MAML notation is wieldy usable (+1.17 compared to IFML), fast to learn (+1.17), and self-descriptive (+1.00). All three aspects relate to the ability to understand and apply the notation in modeling tasks. For technical users, the largest differences are found in the self-learnability (+1.05), perceived consistency (+0.80), and pace of learning (+0.70), also indicating that the process-oriented modeling approach does not harm their technical understanding but is less essential than the emphasis on correctly applying the notation as expected in RQ3.

Regarding app generation, all steps can be executed without additional configuration. The intermediate transformation step is however no inherent limitation of the framework. Future generators targeting new platforms may just as well generate code based on MAML models directly. Apart from reusing existing MD^2 generators developed in the last years, the intermediate transformation adds a technical representation with detailed possibilities of configuration such as UI element styling [4]. Users accustomed to MD^2 may therefore adapt the default representations in the created textual model according to their needs.

The usability study also revealed potentials for further refinement: Although modeling activities were mostly performed correctly, the lack of a screen-oriented representation similar to IFML's nested element structure was mentioned by four participants. A generic and read-only preview of a possible outcome on screen would improve this issue. Also, the wording of some elements proved to be not clear enough. For instance, some data type names are hard to understand for domain experts without further explanation and may be replaced by symbols for a more intuitive understanding.

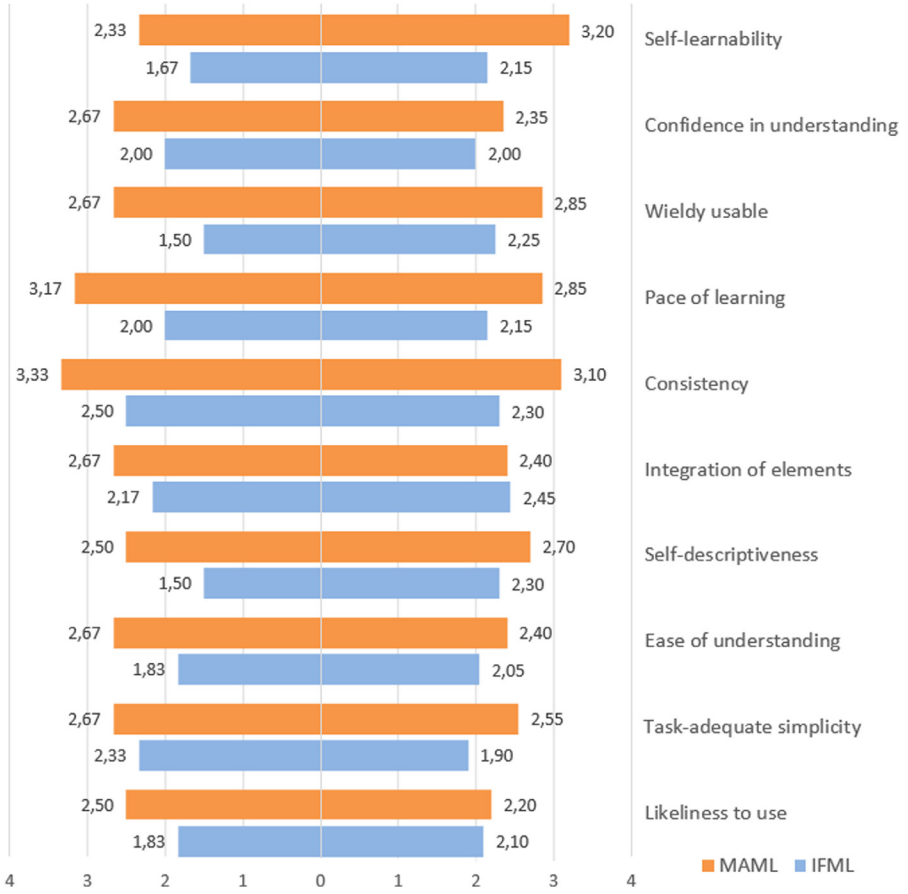

Fig. 16. SUS answers for domain experts (left) and technical users (right).

5. Conclusion and outlook

In this article, the MAML framework was proposed to model mobile apps using a declarative graphical DSL. In contrast to the current practice of graphically configuring user interfaces by positioning UI elements on a screen-like canvas, MAML focuses on a process-centric definition of business apps. Using a sequence of platform-agnostic process elements, the notation aligns with the business perspective of managing processes and data flows, and makes app development accessible to domain users without software engineering experience. The approach is based on existing work on cross-platform business app generation and uses model-driven techniques to transform the models first into an intermediate textual representation before generating platform-specific source code. In particular, a data model inference mechanism was presented that enables real-time validation and consistency checks on partial data models, and overcomes the need for explicitly modeling a global data schema. Moreover, the inferred data can be used to provide contextual modeling support and enhanced semantic validation in the editor component. An empirical evaluation study supports the advantage of MAML over the related technical IFML notation, specifically with regard to its readability by domain experts. MAML therefore achieves the desired balance of abstracting programming-heavy tasks to understandable process flows while keeping the technical expressiveness required for automatic source code generation for multiple target platforms.

The study results support the benefit of MAML and the participants can be seen as realistic sample for app-experienced adults in the general workforce. Still, the amount of student participants may pose a threat to validity and more extensive studies are necessary to confirm these results. Applying the prototype to real-world use cases might reveal further need for improvements and at the same time constitutes future work. Programmers may want to deviate from the default configuration used for automatic app generation, for example by performing manual changes to the intermediate representation. Further research needs to be done on how to avoid interference of generated and custom DSL content in the context of iterative development with frequent re-generation cycles.

Finally, the principles of MAML can be applied to mobile devices beyond smartphones and tablets. Regarding the emergence of novel app-enabled devices such as smartwatches, interesting questions arise on transferring model-driven development approaches to different device classes. Ideally, best practices can be found for reusing the same input models for generating mobile business apps on heterogeneous devices with different capabilities and user interaction patterns.

References

[1] 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.
[2] Palmieri M, Singh I, Cicchetti A. Comparison of cross-platform mobile development tools. In: Proceedings of the 16th international conference on intelligence in next generation networks (ICIN); 2012. p. 179–86. doi:10.1109/ICIN.2012.6376023.
[3] Charkaoui S, Adraoui Z, Benlahmar EH. Cross-platform mobile development approaches. In: Proceedings of the colloquium in information science and technology, CIST; 2014. p. 188–91. doi:10.1109/CIST.2014.7016616.
[4] Majchrzak TA, Ernsting J, Kuchen H. Achieving business practicability of model-driven cross-platform apps. OJIS 2015;2(2):3–14. doi:10.19210/OJIS_2015v2i2n02_Majchrzak.
[5] Esperalta D. Decsoft - App Builder. 2017. https://www.davidesperalta.com/appbuilder.
[6] Xamarin Inc. Developer center - xamarin. 2017. https://developer.xamarin.com.
[7] Mernik M, Heering J, Sloane AM. When and how to develop domain-specific languages. ACM Comput Surv 2005;37(4):316–44. doi:10.1145/1118890.1118892.
[8] Kosar T, Mernik M, Carver JC. Program comprehension of domain-specific and general-purpose languages: comparison using a family of experiments. Emp Softw Eng 2012;17(3):276–304. doi:10.1007/s10664-011-9172-x.
[9] Hemel Z, Visser E. Declaratively programming the mobile web with Mobl. In: Proceedings of the international conference on object oriented programming systems languages and applications. OOPSLA. ACM; 2011. p. 695–712. ISBN 978-1-4503-0940-0. doi:10.1145/2048066.2048121.
[10] Jones C, Jia X. Using a domain specific language for lightweight model-driven development. In: Proceedings of the ENASE 2014; 2015. p. 46–62. ISBN 978-3-642-23390-6. doi:10.1007/978-3-642-23391-3.
[11] Heitkötter H, Majchrzak TA. Cross-platform development of business apps with MD^2. In: Proceedings of the international conference on design science at the intersection of physical and virtual design (DESRIST). In: *LNBIP*, vol. 7939. Springer; 2013. p. 405–11. doi:10.1007/978-3-642-38827-9_29.
[12] Umuhoza E, Brambilla M. Model driven development approaches for mobile applications: a survey. In: Younas M, Awan I, Kryvinska N, Strauss C, van Thanh D, editors. Mobile web and intelligent information systems (MobiWIS). Springer; 2016. p. 93–107. ISBN 978-3-319-44215-0. doi:10.1007/978-3-319-44215-0_8.
[13] Zdun U, Strembeck M. Reusable architectural decisions for DSL design: foundational decisions in DSL development. In: Proceedings of the European conference on pattern languages of programs (EuroPLoP); 2009. p. 1–37.
[14] Breu R, Kuntzmann-Combelles A, Felderer M. New perspectives on software quality. IEEE Softw. 2014;31(1):32–8. doi:10.1109/MS.2014.9.
[15] Meliá S, Cachero C, Hermida JM, Aparicio E. Comparison of a textual versus a graphical notation for the maintainability of mde domain models: An empirical pilot study. Softw. Qual. J. 2016;24(3):709–35. doi:10.1007/s11219-015-9299-x.
[16] Object Management Group. Business process model and notation 2.0, 2011.
[17] Object Management Group. Interaction flow modeling language 1.0, 2015.
[18] Żyła K. Perspectives of simplified graphical domain-specific languages as communication tools in developing mobile systems for reporting life-threatening situations. Stud Logic, Grammar Rhetoric 2015;43(1). doi:10.1515/slgr-2015-0048.
[19] Safdar SA, Iqbal MZ, Khan MU. Empirical evaluation of UML modeling tools–a controlled experiment. In: Taentzer G, Bordeleau F, editors. Modelling foundations and applications. *Lecture Notes in Computer Science*, vol. 9153. Springer; 2015. p. 33–44. ISBN 978-3-319-21150-3. doi:10.1007/978-3-319-21151-0_3.
[20] Buchmann T. Valkyrie: a UML-based model-driven environment for model-driven software engineering. In: Proceedings of the international conference on software paradigm trends (ICSOFT); 2012. p. 147–57. doi:10.5220/0004027401470157.
[21] El Kouhen A, Dumoulin C, Gerard S, Boulet P. Evaluation of modeling tools adaptation. 2012. https://hal.archives-ouvertes.fr/hal-00706701.
[22] Kolovos DS, García-Domínguez A, Rose LM, Paige RF. Eugenia: towards disciplined and automated development of GMF-based graphical model editors. Softw Syst Model 2017;16(1):229–55. doi:10.1007/s10270-015-0455-3.
[23] Rieger C. Business apps with MAML: a model-driven approach to process-oriented mobile app development. In: Proceedings of the symposium on applied computing. SAC. ACM; 2017. p. 1599–606. ISBN 978-1-4503-4486-9. doi:10.1145/3019612.3019746.
[24] El-Kassas WS, Abdullah BA, Yousef AH, Wahba AM. Taxonomy of cross-platform mobile applications development approaches. Ain Shams Eng. J. 2015. doi:10.1016/j.asej.2015.08.004.
[25] Chadha S, Byalik A, Tilevich E, Rozovskaya A. Facilitating the development of cross-platform software via automated code synthesis from web-based programming resources. Comput Lang Syst Struct 2017;48:3–19. doi:10.1016/j.cl.2016.08.005.
[26] Apache Software Foundation. Apache cordova documentation. 2016. https://cordova.apache.org/docs/en/latest/guide/overview/.
[27] Vaupel S, Taentzer G, Harries JP, Stroh R, Gerlach R, Guckert M. Model-driven development of mobile applications allowing role-driven variants. In: Lecture notes in computer science, 8767; 2014. p. 1–17.
[28] Majchrzak TA, Ernsting J. Reengineering an approach to model-driven development of business apps. In: Proceedings of the SIGSAND/PLAIS EuroSymposium; 2015. p. 15–31. doi:10.1007/978-3-319-24366-5_2.
[29] Ernsting J, Rieger C, Wrede F, Majchrzak TA. Refining a reference architecture for model-driven business apps. In: Proceedings of the international conference on web information systems and technologies (WEBIST); 2016. p. 307–16. doi:10.5220/0005862103070316.
[30] Rouly JM, Orbeck JD, Syriani E. Usability and suitability survey of features in visual IDEs for non-programmers. In: Proceedings of the workshop on evaluation and usability of programming languages and tools. PLATEAU. ACM; 2014. p. 31–42. ISBN 978-1-4503-2277-5. doi:10.1145/2688204.2688207.
[31] Namoun A, Daskalopoulou A, Mehandjiev N, Xun Z. Exploring mobile end user development: existing use and design factors. IEEE Trans Softw Eng 2016;42(10):960–76. doi:10.1109/TSE.2016.2532873.
[32] Bubble Group. Bubble - visual programming. 2016. https://bubble.is/.
[33] Acerbis R, Bongio A, Butti S, Brambilla M. Model-driven development of cross-platform mobile applications with webratio and IFML. In: Proceedings of the international conference on mobile software engineering and systems. MOBILESoft. IEEE; 2015. p. 170–1. ISBN 978-1-4799-1934-5. doi:10.1109/MobileSoft.2015.49.
[34] Bizness Apps. Mobile app maker – bizness apps. 2016. http://biznessapps.com/.
[35] Knuplesch D, Reichert M, Ly LT, Kumar A, Rinderle-Ma S. Visual modeling of business process compliance rules with the support of multiple perspectives. In: Ng W, Storey VC, Trujillo JC, editors. Conceptual modeling (ER). Springer; 2013. p. 106–20. ISBN 978-3-642-41924-9. doi:10.1007/978-3-642-41924-9_10.
[36] Hitachi Vantara Corp. Data integration - Kettle. 2017. http://pentaho.com/product/data-integration.
[37] Wolber D. App inventor and real-world motivation. In: Proceedings of the ACM technical symposium on computer science education (SIGCSE); 2011.
[38] Danado J, Paternò F. Puzzle: a visual-based environment for end user development in touch-based mobile phones. In: Winckler M, Forbrig P, Bernhaupt R, editors. Human-centered software engineering (HCSE). Springer; 2012. p. 199–216. ISBN 978-3-642-34347-6. doi:10.1007/978-3-642-34347-6_12.
[39] Barnett S, Avazpour I, Vasa R, Grundy J. A multi-view framework for generating mobile apps. In: Proceedings of the IEEE symposium on visual languages and human-centric computing (VL/HCC); 2015. p. 305–6. doi:10.1109/VLHCC.2015.7357239.
[40] Object Management Group. Unified modeling language 2.5. 2015.
[41] van der Aalst W. Formalization and verification of event-driven process chains. Inf Softw Technol 1999;41(10):639–50. doi:10.1016/S0950-5849(99)00016-6.
[42] van der Aalst W, ter Hofstede A. YAWL: Yet another workflow language. Inf Syst 2005;30(4):245–75. doi:10.1016/j.is.2004.02.002.
[43] International Organization for Standardization. ISO 5807:1985. 1985.

[44] Schalles C. Usability evaluation of modeling languages. Springer Fachmedien Wiesbaden; 2013. doi:10.1007/978-3-658-00051-6.
[45] Challenger M, Erata F, Onat M, Gezgen H, Kardas G, Herbstritt M, editors. A model-driven engineering technique for developing composite content applications. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik GmbH, Wadern/Saarbruecken, Germany; 2016. doi:10.4230/OASIcs.SLATE.2016.11.
[46] Franzago M, Muccini H, Malavolta I. Towards a collaborative framework for the design and development of data-intensive mobile applications. In: Proceedings of the international conference on mobile software engineering and systems. MOBILESoft. ACM; 2014. p. 58–61. ISBN 978-1-4503-2878-4. doi:10.1145/2593902.2593917.
[47] Koch N, Knapp A, Zhang G, Baumeister H. UML-based web engineering. In: Rossi G, Pastor O, Schwabe D, Olsina L, editors. Web engineering: modelling and implementing web applications. Human-Computer Interaction Series. Springer; 2008. p. 157–91. ISBN 978-1-84628-922-4. doi:10.1007/978-1-84628-923-1_7.
[48] Francese R, Risi M, Scanniello G, Tortora G. Model-driven development for multi-platform mobile applications. In: Abrahamsson P, Corral L, Oivo M, Russo B, editors. Product-focused software process improvement (PROFES). Springer; 2015. p. 61–7. ISBN 978-3-319-26844-6. doi:10.1007/978-3-319-26844-6_5.
[49] Simons C, Wirtz G. Modeling context in mobile distributed systems with the UML. J Visual Lang Comput 2007;18(4):420–39. doi:10.1016/j.jvlc.2007.07.001.
[50] Brambilla M, Mauri A, Umuhoza E. Extending the interaction flow modeling language (IFML) for model driven development of mobile applications front end. In: Lecture notes in computer science, 8640; 2014. p. 176–91. doi:10.1007/978-3-319-10359-4_15.
[51] Brambilla M, Dosmi M, Fraternali P. Model-driven engineering of service orchestrations. In: Proceedings of the world congress on services (SERVICES); 2009.
[52] France RB, Ghosh S, Dinh-Trong T, Solberg A. Model-driven development using UML 2.0: promises and pitfalls. Computer 2006;39(2):59–66. doi:10.1109/MC.2006.65.
[53] Granada D, Vara JM, Brambilla M, Bollati V, Marcos E. Analysing the cognitive effectiveness of the webml visual notation. Softw. Syst. Model. 2015. doi:10.1007/s10270-014-0447-8.
[54] Trætteberg H, Krogstie J. Enhancing the usability of BPM-solutions by combining process and user-interface modelling. In: The practice of enterprise modeling: first IFIP WG 8.1 working conference (PoEM). Springer; 2008. p. 86–97. ISBN 978-3-540-89218-2. doi:10.1007/978-3-540-89218-2_7.
[55] Sutanta E, Wardoyo R, Mustofa K, Winarko E. Survey: models and prototypes of schema matching. Int J Elect Comput Eng 2016;6(3):1011–22. doi:10.11591/ijece.v6i3.9789.
[56] Enjo H, Iijima J. Towards class diagram algebra for composing data models. In: Proceedings of the conference on new trends in software methodologies, tools and techniques (SoMeT). IOS Press; 2010. p. 112–33. ISBN 978-1-60750-628-7.
[57] Rahm E, Bernstein PA. A survey of approaches to automatic schema matching. VLDB J 2001;10(4):334–50. doi:10.1007/s007780100057.
[58] Lin Y, Gray J, Jouault F. DSMDiff: a differentiation tool for domain-specific models. Eur J Inf Syst 2007;16(4):349–61. doi:10.1057/palgrave.ejis.3000685.
[59] Cappiello C, Matera M, Picozzi M, Caio A, Guevara MT. MobiMash: End user development for mobile mashups. In: Proceedings of the annual conference on world wide web companion (WWW); 2012. p. 473–4. doi:10.1145/2187980.2188083.
[60] Liu Q, Gray J, Mernik M, Bryant BR. Application of metamodel inference with large-scale metamodels. Int J Softw Inf 2012;6(2):201–31.
[61] Javed F, Mernik M, Gray J, Bryant BR. Mars: A metamodel recovery system using grammar inference. Inf Softw Technol 2008;50(9–10):948–68. doi:10.1016/j.infsof.2007.08.003.
[62] López-Fernández JJ, Cuadrado JS, Guerra E, de Lara J. Example-driven meta-model development. Softw Syst Model 2015;14(4):1323–47. doi:10.1007/s10270-013-0392-y.
[63] Rieger C. MAML respository, 2017. https://github.com/wwu-pi/maml.
[64] Gamma E, Helm R, Johnson R, Vlissides J. Design patterns: elements of reusable object-oriented software. Addison-Wesley professional computing series. Addison-Wesley; 1995. ISBN 9780201633610.
[65] Rieger C. A data model inference algorithm for schemaless process modelling. In: Becker J., Backhaus K., Dugas M., Hellingrath B., Hoeren T., Klein S., et al., editors. Working Papers, European Research Center for Information Systems No. 29. ERCIS, University of Münster; 2016, p. 1–17.
[66] The Eclipse Foundation. Sirius. 2016. https://eclipse.org/sirius/.
[67] Smuts M, Burger C, Scholtz B. Composite, real-time validation for business process modelling. In: de Villiers C, van der Merwe AJ, van Deventer JP, Matthee MC, Gelderblom H, Gerber A, editors. Proceedings of the Southern African institute for computer scientist and information technologists annual conference; 2014. p. 93–103. doi:10.1145/2664591.2664603.
[68] Erdweg S, Van Der Storm T, Völter M, Boersma M, Bosman R, Cook W, et al. The state of the art in language workbenches: Conclusions from the language workbench challenge. In: Lecture notes in computer science, 8225 LNCS; 2013. p. 197–217. doi:10.1007/978-3-319-02654-1_11.
[69] Bettini L. Implementing domain-specific languages with Xtext and Xtend. Community experience distilled. Birmingham, UK: Packt Pub; 2013. ISBN 1782160310.
[70] Zhao X, Xia X, Kochhar P, Lo D, Li S. An empirical study of bugs in build process. In: Proceedings of the ACM symposium on applied computing; 2014. p. 1187–9. doi:10.1145/2554850.2555142.
[71] Gujral S, Sharma G, Sharma S, Diksha. Classifying bug severity using dictionary based approach. In: Proceedings of the international conference on futuristic trends on computational analysis and knowledge management (ABLAZE); 2015. p. 599–602. doi:10.1109/ABLAZE.2015.7154933.
[72] The Eclipse Foundation. Model-to-model transformation. The Eclipse Foundation2016. https://projects.eclipse.org/projects/modeling.mmt.
[73] Rieger C, Majchrzak TA. Conquering the mobile device jungle: towards a taxonomy for app-enabled devices. In: Proceedings of the international conference on web information systems and technologies (WEBIST); 2017. p. 332–9. ISBN 978-989-758-246-2.
[74] Sanaei Z, Abolfazli S, Gani A, Buyya R. Heterogeneity in mobile cloud computing: taxonomy and open challenges. IEEE Commun Surv Tutor 2014;16(1):369–92. doi:10.1109/SURV.2013.050113.00090.
[75] Martínez JS, García-Bustelo BCP, Díaz VG, Lovelle JMC. Isastur modeler: a tool for BPMN MUSIM. In: Proceedings of the Iberian conference on information systems and technologies (CISTI); 2011. p. 1–6.
[76] Brooke J. SUS - a quick and dirty usability scale. Usability Evaluat Ind 1996;189(194):4–7.
[77] Bangor A, Kortum P, Miller J. Determining what individual SUS scores mean: adding an adjective rating scale. J Usability Stud 2009;4(3):114–23.
[78] Petre M. UML in practice. In: Proceedings of the international conference on software engineering. ICSE. IEEE; 2013. p. 722–31. ISBN 978-1-4673-3076-3. doi:10.1109/ICSE.2013.6606618.
[79] Noy NF. Semantic integration: a survey of ontology-based approaches. SIGMOD Record 2004;33(4):65–70. doi:10.1145/1041410.1041421.