Working Papers

ERCIS — European Research Center for Information Systems
Editors: J. Becker, K. Backhaus, M. Dugas, B. Hellingrath,
T. Hoeren, S. Klein, H. Kuchen, U. Müller-Funk, H. Trautmann, G. Vossen

Working Paper No. 29

A Data Model Inference Algorithm for Schemaless Process Modeling

Christoph Rieger

ISSN 1614-7448

cite as: Christoph Rieger: A Data Model Inference Algorithm for Schemaless Process Modeling. In: Working Papers, European Research Center for Information Systems No. 29. Eds.: Becker, J. et al. Münster 2016.

Contents

List of Figures

Working Paper Sketch

Type

Research Report

Title

A Data Model Inference Algorithm for Schemaless Process Modeling.

Authors

Christoph Rieger

contact via: christoph.rieger@uni-muenster.de

Abstract

Mobile devices have become ubiquitous not only in the consumer domain but also support the digitalization of business operations though business apps. Many frameworks for *programming* cross-platform apps have been proposed, but only few *modeling* approaches exist that focus on platform-agnostic representations of mobile apps. In addition, app development activities are almost exclusively performed by software developers, while domain experts are rarely involved in the actual app creation beyond requirements engineering phases.

This work concentrates on a model-driven approach to app development that is also comprehensible to non-technical users. With the help of a graphical domain-specific language, data model, view representation, business logic, and user interactions are modeled in a common model from a process perspective. To enable such an approach from a technical point of view, an inference mechanism is presented that merges multiple partial data models into a global specification. Through model transformations, native business apps can then be generated for multiple platforms without manual programming.

Keywords

Graphical DSL, Mobile Application, Business App, Model-driven software development, Data model inference

1 Introduction

Mobile devices have become ubiquitous not only in the consumer domain but also support the digitization of business operations [37]. In particular since the advent of Apple's first iPhone in 2007 [2], a trend towards small-scale applications for specific use cases, so-called apps, has emerged. Whereas apps are now *used* in various consumer and business contexts, app *development* is still a task exclusively executed by programmers. Other stakeholders and future users are involved primarily in requirements engineering phases upfront implementation, following established software engineering methodologies.

However, Gartner predicts that in the next two years, more than half of all business apps for company-internal purposes will be created using codeless tools [37]. One approach to codeless app creation is model-driven software development. Modeling such apps can be achieved using two kinds of notations: On the one hand, a wide variety of general purpose process modeling notations such as Business Process Model and Notation (BPMN) or Event-driven Process Chains (EPC) exists [32, 42]. Usually, those models cannot be directly transformed into mobile apps because of lacking mobile-specific details and semantics. From a technical perspective, these notations often just represent connected, non-interpretable boxes filled with text. On the other hand, technical notations such as the Interaction Flow Modeling Language (IFML) are too complex to understand for domain experts and require software engineering knowledge [33, 45]. In addition, to adequately model mobile apps, a suitable notation needs to be platform-independent to cover the variety of platforms and device types.

Many frameworks for *programming* cross-platform apps have been proposed [12, 10], but only few *modeling* approaches exist that focus on platform-agnostic representations of mobile apps. Existing commercial platforms provide cross-platform capabilities, but usually limited to source code transformations or partly supported by graphical editors for designing individual views (e.g. [15]). Model-driven software development [39] instead utilizes app models as input for a partly or fully automated creation of apps. Various textual domain-specific languages (DSL) [31] such as Mobl, AXIOM, and MD^2 follow this approach [22, 24, 21]. DSLs are suited to cover a well-defined scope, the so-called domain, and model inherent domain concepts on a more abstract level. However, a textual representation provides only minor benefits to non-technical users. Textual modeling is potentially more concise but still feels like programming [44]. Yet, input from stakeholders with strong domain knowledge is essential to ensure the developed software matches their tacit requirements [7].

The Münster App Modeling Language (MAML; pronounced 'mammal') framework aims to alleviate the aforementioned problems. This paper's contributions are twofold: First, a model-driven approach to app development is presented which is also comprehensible for non-technical users. With the help of a graphical domain-specific language, data model, view representation, business logic, and user interactions are defined in a common model from a process perspective. Through model transformations, native business apps can then be generated for multiple platforms without manual programming. Second, to enable such an approach from a technical point of view, an inference mechanism is required that merges multiple partial data models into a global specification. Consequently, the modeler is disburdened from explicit data model specifications that need to be maintained separately and require knowledge about the application as a whole.

The structure of the paper follows these contributions. After presenting related work in Section 2, the proposed framework is presented in Section 3. Section 4 explains how to infer a data model from a set of MAML models. Finally, this report concludes in Section 5 and gives an outlook on future work.

2 Related work

The work presented in this paper draws on the scientific fields of cross-platform mobile app generation, domain-specific visual languages, and schema matching. Regarding cross-platform mobile apps, different approaches exist that can be grouped into three major categories according to El-Kassas [12]: Existing source code can be *compiled* from a legacy application or different platform, a runtime or virtual machine can *interpret* a common code base, or app source code can be generated in a *model-driven* way from a textual or graphical representation. With regard to model-driven approaches, a variety of frameworks can be found in academic literature [41]. Only few of them, such as Mobl [22] and AXIOM [24], set out to cover the full spectrum of structure and runtime behavior of an app; often providing a custom textual DSL for this means. This paper builds on previous work on the MD^2 framework which focuses on the generation of business apps, i.e., form-based, data-driven apps interacting with back-end systems according to Majchrzak et al. [30]. The input model is specified using a platform-independent, textual DSL [21]. After preprocessing, code generators transform it into platform-native source code [29, 14]. Despite several development-related improvements in the past years, textual DSLs are generally often perceived as programming to non-technical users [45]. This barrier to a wide-spread usage of textual DSLs motivates the need for further abstraction and visual modeling.

Graphical DSLs or visual programming languages exist for several purposes, including process-related domains such as business process compliance [25] and data integration [35]. Regarding mobile applications, RAPPT represents a model-driven approach that mixes a graphical DSL for process flows with a textual DSL for programming [3]; AppInventor encourages novices to create apps by combining building blocks of a visual programming language [43]. Puzzle takes development of mobile applications one step further by providing a visual development environment on the mobile device itself [11]. Although all of these approaches aim at simplifying the actual programming tasks for novice developers, they disregard non-technical stakeholders.

In contrast, general purpose modeling notations exist to describe applications and process flows. The Unified Modeling Language (UML) for the domain of software development provides several standards to define the structure and runtime behavior of an application [34]. One of them, IFML (succeeding WebML), specifies user interactions within a software system [33]. Cognitive studies have been conducted, e.g. for WebML, and showed that technical modeling notations are often considered as complex to understand for domain experts. Problems such as symbol overload are further aggravated by the combined use of multiple notations in order to describe all behavioral and structural characteristics [20, 16]. To visualize process flows in general, a variety of modeling notations exist, e.g. BPMN, EPC, or flowcharts [32, 42, 23]. However, such notations are often superficial regarding technical specificity, and mobile-related aspects are often not included because of their general applicability. Modeling approaches specific to the mobile domain rarely reach beyond user interface modeling. Some approaches explicitly try to incorporate non-technical users, e.g., through collaborative multi-viewpoint modeling [18]. Others use existing modeling notations such as statecharts [17] or extend them for mobile purposes, e.g. UML to model context in mobile distributed systems [38], IFML with mobile-specific elements [6], or BPMN to orchestrate web services [5].

Schema matching and model differentiation are further relevant fields of research with various approaches regarding the identification of common structures in models [40]. According to the classification by Rahm [36], a schema-only and constraint-based approach on element level is required for the inference of a global data model from multiple input models. For the given problem of partial models, inference can be limited to additive and name-based approaches. Enjo and Iijima presented related work on the UML composition of class diagrams [13]. More sophisticated strategies, e.g. relying on ontologies, might also identify modeling inconsistencies beyond strict name-based matching [26]. An application related to mobile devices, but focused on the visualization of data instead of its manipulation, is MobiMash for graphically creating mashup apps by configuring the representation and orchestration of data services [9].

In the context of meta modeling, reverse engineering approaches to track meta model evolution deal with similar problems of inferring object structures [27], and López-Fernández et al. [28] presented a related idea in order to derive a common meta model from exemplary model fragments.

The commercial WebRatio Mobile Platform is closest to the work presented in this paper as it can also generate apps from a graphically edited model [1]. However, it has a high entry barrier for potential non-technical users by relying on the combination of IFML and additional notations. Also, it does not provide modularized app models with extended mechanisms for inferring data structures as presented in Section 4. With similar ambitions, companies such as BiznessApps and Bubble promise codeless creation of apps using detailed configurators and web-based user interface editors [4, 8]. This work in contrast focuses on a process-centric and platform-agnostic approach as described next, with a higher level of abstraction than simple drag-and-drop configuration tools.

3 Münster App Modeling Language

The MAML DSL is built around five main principles:

■ Domain expert focus: In contrast to technical specification languages, MAML is designed with a non-technical user in mind; regarding both the actual models and the modeling environment. Therefore, process modelers or domain experts without software development experience should be able to understand, create, and modify models without longsome training.

■ Data-driven process: MAML models represent a process perspective on business apps, visualizing the flow of data objects through a sequence of processing steps. Compared to other process modeling notations, the content and structure of these data objects are explicated in the model.

■ Modularization: The scope of a model is one *Use Case*, a unit of useful functionality comprising a self-contained set of behaviors and interactions performed by the app user [34]. To support the domain expert focus, MAML combines data model, process flow, app visualization, and user interactions in a single model. This opposes software engineering patterns such as Model-View-Controller [19] that separate such aspects for better maintainability of large-scale software.

■ Declarative description: Use cases contain platform-agnostic elements describing *what* processing activities are possible on the data objects. However, the concrete representation on a mobile device is not further specified on this abstract level. During the generation phase, sensible defaults for platform-specific appearances are provided.

■ Automatic cross-platform app generation: One major design goal of the MAML framework is its capability to create fully-functional software for multiple platforms in order to reach a large amount of users. The graphical model is therefore designed to be interpretable by different code generators without further need for manual programming. As a result, MAML provides the means for codeless development of business apps.

Figure 1 depicts a sample *use case* for adding an item to a publication management system (enriched with numbered circles for reference in the following paragraphs). The model contains a sequence of activities, from a *start event* (1) towards one or several *end events* (2). In the beginning, a *local* or *remote data source* (3) specifies the data type of the manipulated objects. Figure 1 depicts the data type ”Publication” to be manipulated within the process steps and which is managed by a remote server. The modeler can then choose from a variety of (arrow-shaped) *interaction process elements* (4), for example to *select/create/update/display/delete entities*, show *popup messages*, or access device functionalities such as the *camera* and starting a phone *call*. Due to its declarative nature, the DSL does not indicate the concrete appearance but typically each logical process step may be rendered as one view of the app. Furthermore, *automated process*

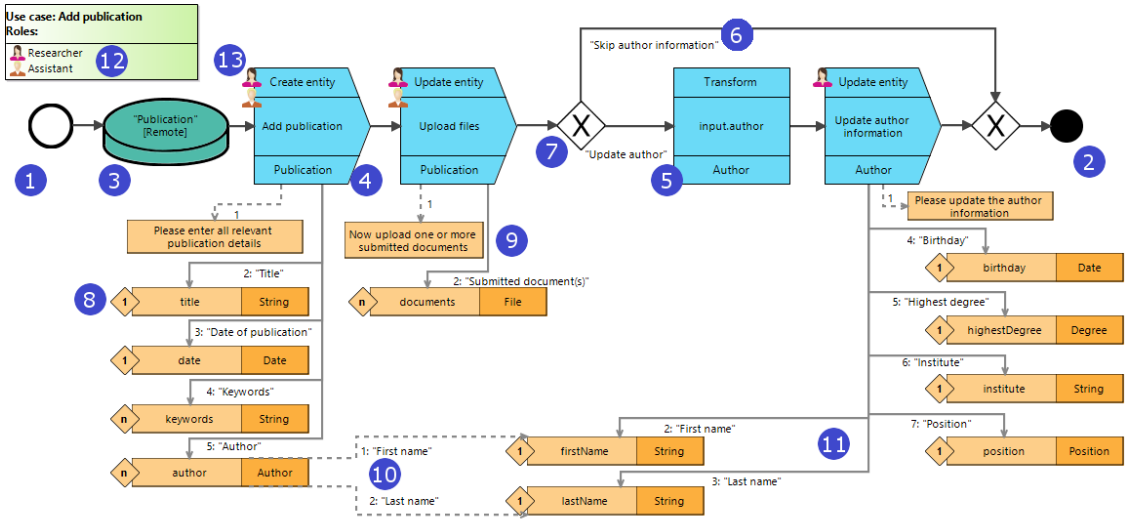

Figure 1: Sample MAML use case “Add publication”

elements (5) represent invisible processing steps without user interaction, e.g. calling RESTful *web services*, *including* other models for process modularization, or navigating through the object structure (*transform*).

The navigation between connected process steps happens using an automatically created “Continue” button. Alternatively, a distinct denomination can be specified along the *process connectors* (6). To allow for conditional actions, the process flow can be branched out using an *XOR* element (7). The condition can either be triggered automatically by attaching an attribute to the XOR element and evaluating specified expressions, or requires a user decision by providing respective button captions along the process connectors (such as in the example). To sum up, the exemplary use case first creates a new entity of type ”Publication”, then edits additional data before optionally traversing the object structure to the publication’s author and editing his/her data.

The rectangular elements (bottom half of Figure 1) represent the data structure which is relevant for a particular process. Every process element needs to specify all (and only those) attributes which are displayed on the screen or utilized in automated processing activities. *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., arbitrary 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*, depicted as rectangle without cardinality and type information (9), can be added to display explanatory text on screen, and *computed attributes* (not illustrated) may be used to output calculations on other attributes at runtime. To assign these UI elements to a process step, two types of connectors exist: Dotted arrows (10) represent a *consuming relationship* whereas solid arrows (11) signify a *modifying relationship* regarding the target element. This refers not only to the manifest representation of attribute content displayed either as read-only text or editable input field. The interpretation also applies in a wider sense, e.g. web service calls in which the server “reads” an input parameter and “modifies” information through its response.

Every connector which is connected to an interaction process element also specifies an order of appearance. Additionally, a human-readable representation of the field description is derived from the attribute name unless specified manually (10). To reduce the amount of elements to be modeled, multiple connectors may point to the same UI element from different sources (given their data types match). Alternatively, to avoid wide-spread connections across larger models, UI

elements may instead be duplicated to different positions in the model and will automatically be recognized as being the same element (see Section 4).

Finally, the MAML DSL supports a multi-role concept. The modeler can specify role names (12) and annotate them to the respective interaction process elements (13). This is particularly useful to describe scenarios in which parts of the process are performed by different people, e.g. approval workflows. If the assigned role changes, the process automatically terminates for the first app user, modified data objects are saved, and the subsequent user is informed about an open workflow instance in his app. The exemplary use case of Figure 1, the publication might be added by a researcher or assistant, however, the author information can only be changed by a researcher.

4 Data model inference algorithm

The most important advantage of MAML's approach is the renunciation of a global data schema that needs to be modeled and maintained separately. Instead, each process step refers (just) to the attributes which are required (i.e. displayed or edited) within this particular step. As result of the modeling activities, only partial models exist that need to be matched on multiple levels: for each process element individually, for the use case as a whole, and across multiple use cases of the overall app.

4.1 Partial data model inference

First, separate data models need to be inferred for each process element. The challenge lies in the unidirectional specification of relationships, i.e. a MAML process element with a given data type "has a" relationship to an *attribute* of a specified type and cardinality but there is mostly no explicit information on the opposite relationship.

Let M denote the set of use case models for which a coherent data model should be inferred. Also, D_m denotes the set of data types within a concrete model $m \in M$. Then,

$$R_m \subset D_m \times String \times D_m \times \mathcal{P}(\mathbb{N}) \times \mathcal{P}(\mathbb{N})$$

denotes the set of *relationship* tuples between two distinct data types. Within such a tuple $r_i = (s_i, rn_i, t_i, sc_i, tc_i)$, the value s_i represents the source data type of a relationship, rn_i the corresponding name, and t_i the target data type. A *cardinality* represents the possible number of values referred to by a relationship [34], using the notation $i..j = \{i, i+1, ..., j\}$ and n represents infinity. sc_i signifies the source cardinality of the relationship and rc_i the respective target cardinality.

Then, the annotated directed graph $G_m := (D_m, R_m)$ represents the data model of m with data types as vertices and relationships as edges. In particular, the graph may contain multiple edges between the same pair of source and target data types, if the relationship names, the source cardinalities, or the target cardinalities differ. Whereas the former is a valid modeling option (e.g., lectures are held by teachers and attended by students which are both of type "person"), the differing cardinalities can be considered as modeling error (cf. Section 4.3).

In addition, let V_m denote the set of primitive types within a concrete model $m \in M$, meaning atomic values which do not contain any relationships to other data types. Then,

$$P_m \subset D_m \times String \times V_m \times \mathcal{P}(\mathbb{N})$$

represents the set of *property* tuples. Accordingly, for a tuple $p_i = (ps_i, pn_i, pt_i, pc_i)$ the value ps_i represents the source data type that contains a property of the primitive type pt_i with the name pn_i and cardinality ps_i.

To apply the above definitions to MAML, a data type is specified either within the data source element, a process element, or as part of an attribute element. For example, Figure 2 depicts the visual representations of a remote data source referring to the data type ”Publication” (a), an update entity interaction process element for the same data type (b) as well as an attribute for the data type ”Author” (c). In addition, MAML provides a pre-defined list of primitive types (integer, floating point number, string, location, boolean, date, time, datetime, and file) from which these data types can be composed.

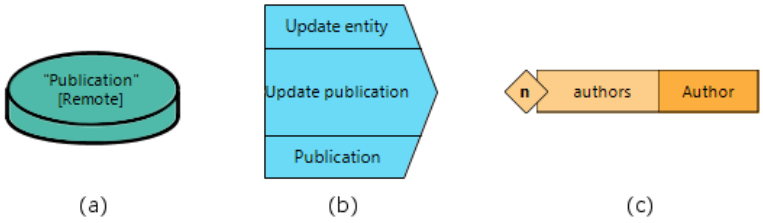

Figure 2: Specification options for data types in MAML

In order to identify interrelations between data types, three origins need to be considered (cf. numbered circles in Figure 3): First, a relationship may exist between a process element and an attached attribute. Second, a nested attribute adds a relationship to the nesting attribute’s data type. Third, an attribute may be transitively connected to the process element through one or more computed attributes (e.g., (3) in Figure 3 representing a ”count” aggregation operator), but still refers to the process element’s data type (unless it has other incoming attribute connectors). MAML does not differentiate between primitive types and data types with regard to their representation in the model. Therefore, the aforementioned options might translate to either a *relationship* or *property* of the originating data type.

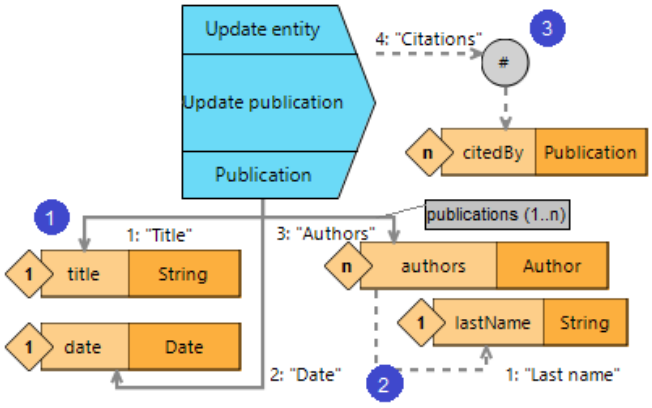

Figure 3: Attribute relation options in MAML models

The cardinalities which can initially be assigned to a relationship depend on the type of association. Four main cases can be distinguished:

■ Primitive: MAML attributes with primitive data types are trivially converted to single- or multi-valued *properties* of the source data type. For example, the connection (2) in Figure 3 translates to the tuple $(Author, ”lastName”, String, 0..1)$ in P_m [1].

[1] In MAML models, **1** specifies a $0..1$ cardinality and **n** refers to $0..n$.

■ Unidirectional: Typically, modeled *relationships* are unidirectional and can therefore be modeled only in one direction. Each relationship explicitly specifies a name and cardinality. Because multiple objects of the source data type might reference the same target object, the unspecified source cardinality is unknown and must be interpreted as unrestricted with $0..n$. In the example, the connection (3) translates to the tuple $(Publication, "citedBy", Publication, 0..n, 0..n)$ in R_m.

■ Bidirectional: In contrast to the previous case, bidirectional relationships between data types d_1 and d_2 are fully specified in the graphical model and provide both source and target cardinalities. In MAML, this is represented by an additional annotation (containing the name and cardinality for the opposite direction) along the connecting arrow, e.g., the ”authors” relationship in Figure 3. To capture the navigability in both directions – and particularly the respective attribute names – in the graph, a second relationship is inserted with inverted order of data types and cardinalities.

■ Singleton: Relationships originating from singleton data types are a variant of the unidirectional scenario in which the unknown cardinality can be restricted to 0..1 (a maximum of one object can be set) [19]. In MAML, singleton data types are created when using the singleton data source element within the process flow.

Two models are considered *compatible*, if the combined constraints of both models for data type, name, and cardinality consistency are satisfiable (cf. Section 4.3). As an example, Figure 4 depicts two compatible MAML models and Listing 1 shows the corresponding graph structure.

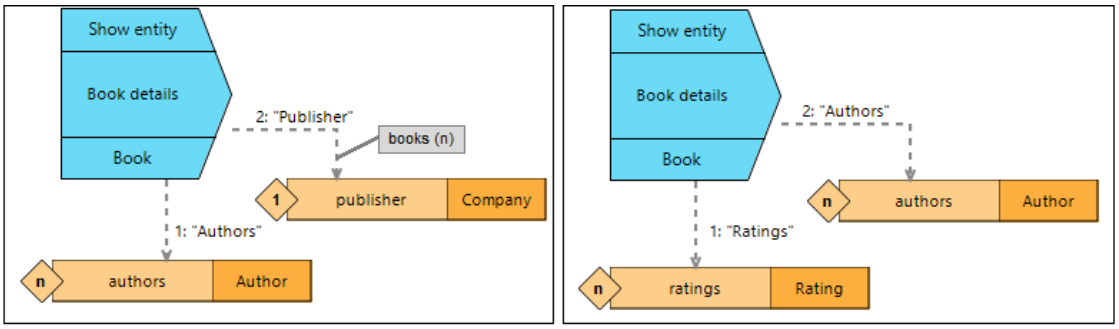

Figure 4: Compatible partial MAML models

Listing 1 Exemplary data structure for two compatible models

$M = \{m_1, m_2\}$
$D_{m_1} = \{Book, Author, Company\}$ ▻ Data types
$D_{m_2} = \{Book, Rating, Author\}$
$V_{m_1} = V_{m_2} = \varnothing$

$P_{m_1} = P_{m_2} = \varnothing$ ▻ Properties

$r_1 = (Book, "authors", Author, 0..n, 0..n)$ ▻ Relationships
$r_2 = (Book, "publisher", Company, 0..n, 0..1)$
$r_3 = (Company, "books", Book, 0..1, 0..n)$
$R_{m_1} = \{r_1, r_2, r_3\}$

$r_4 = (Book, "ratings", Rating, 0..n, 0..n)$
$r_5 = (Book, "authors", Author, 0..n, 0..n)$
$R_{m_2} = \{r_4, r_5\}$

4.2 Merging partial data models

By using an associative and commutative merging operation, all partial models can be merged iteratively or simultaneously into a single global data model G_g before identifying modeling inconsistencies as described in Section 4.3.

First, all distinct data types and primitive types can be aggregated from the considered source models directly:

$$D_g = \bigcup_{m \in M} D_m \tag{1}$$

$$V_g = \bigcup_{m \in M} V_m \tag{2}$$

Second, relationships from each model m are added to R_g if there is yet no relationship between both data types, or (given that source and target data type match) the name or source cardinality, or target cardinality of the relationship differs. Due to the representation of R_m as set of tuples, this boils down to the union of all relationship sets of the source models:

$$R_g = \bigcup_{m \in M} R_m \tag{3}$$

Properties of the source models are likewise merged:

$$P_g = \bigcup_{m \in M} P_m \tag{4}$$

Applied to the example of Figure 4, the relationship r_5 is equivalent to r_1 and therefore ignored. The resulting graph structure is depicted in Listing 2 and the corresponding UML class diagram in Figure 5.

Listing 2 Exemplary merged data structure of two compatible models

$D_g = \{Book, Author, Company, Rating\}$
$V_g = \varnothing$

$P_g = \varnothing$
$r_1 = (Book, "authors", Author, 0..n, 0..n)$
$r_2 = (Book, "publisher", Company, 0..n, 0..1)$
$r_3 = (Company, "books", Book, 0..1, 0..n)$
$r_4 = (Book, "ratings", Rating, 0..n, 0..n)$
$R_g = \{r_1, r_2, r_3, r_4\}$

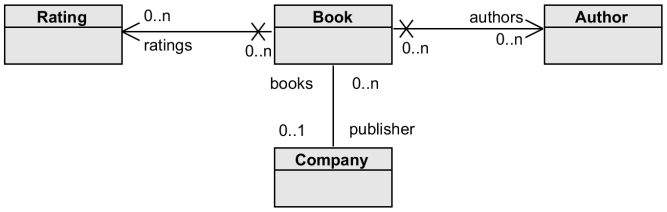

Figure 5: UML class diagram of the merged model

4.3 Consolidation and error identification

Whereas the previous merging step has aggregated all partial models, the resulting global data model might be invalid. For example, both partial models in Figure 6 are valid on their own and can be merged according to equations (1) to (4). However, they are called *conflicting* because merging those models results in a global model that is semantically invalid. Generally, two types of modeling errors can be observed:

■ First, a *type error* exists if any source data type in the graph has two properties or relationships of the same name pointing to different target data types, i.e. not conforming to

$$\forall r_i, r_j \in R_g \mid s_i = s_j,\ rn_i = rn_j : \quad t_i = t_j \tag{5}$$

$$\forall p_i, p_j \in P_g \mid ps_i = ps_j,\ pn_i = pn_j : \quad pt_i = pt_j \tag{6}$$

■ Second, a *name conflict* exists if the same name is assigned more than once to relationships and properties for the same source data type, violating

$$\forall r_i \in R_g, p_j \in P_g \mid s_i = ps_j : \quad rn_i \neq pn_j \tag{7}$$

■ Third, a *cardinality conflict* exists if two *relationships* with the same name differ with regard to their cardinalities for any pair of data types, i.e. violating any of

$$\forall r_i, r_j \in R_g \mid s_i = s_j,\ rn_i = rn_j,\ t_i = t_j : \quad sc_i = sc_j \tag{8}$$

$$\forall r_i, r_j \in R_g \mid s_i = s_j,\ rn_i = rn_j,\ t_i = t_j : \quad tc_i = tc_j \tag{9}$$

A cardinality conflict also exists if two *properties* with the same name differ with regard to their cardinalities for any pair of primitive types, i.e. not conforming to

$$\forall p_i, p_j \in P_g \mid ps_i = ps_j,\ pn_i = pn_j,\ pt_i = pt_j : \quad pc_i = pc_j \tag{10}$$

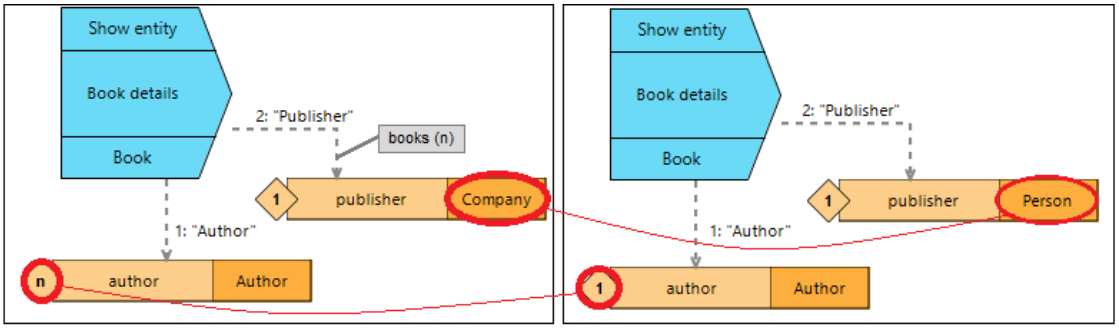

Figure 6: Conflicting partial MAML models

Type errors and name conflicts according to (5) to (7) cannot be resolved automatically and need to be corrected by the modeler. For instance in Figure 6, the inference mechanism cannot decide whether the ambiguous target data type of the ”publisher” attribute should be set to ”Company” or ”Person”.

In case of cardinality conflicts violating (8) to (10), the modeler should be warned, but automatic resolution is possible. For each pair of relationships $r_i, r_j \in R_g$ with matching source data type, target data type, and name, the cardinality for each side of the association can be calculated as

the intersection between the conflicting cardinalities. The cardinality is calculated accordingly for each pair of *properties* $p_i, p_j \in P_g$ with matching source data type, target data type, and name. For the example of Figure 6, the target cardinality $0..n \cap 0..1 = 0..1$ is assigned to the ”author” attribute. Although bidirectional relationships are modeled as two separate tuples, this causes no harm because the resolved cardinality satisfies all constraints.

As a result, the inference algorithm can be applied to partial models of different granularity (first within a single MAML model, then across models) in arbitrary order, and can both serve to validate model correctness and derive a global data model required for software generation and database schema creation.

5 Conclusion and Outlook

In this work, the MAML framework was introduced to alleviate the problems of programmer-focused mobile app development. In future, apps can instead be modeled codelessly by domain experts using a declarative graphical DSL. However, as opposed to visual configuration tools or low-level GUI editors to specify the position of user interface elements on a screen canvas, MAML focuses on a process-centric definition of apps. Using abstract and platform-agnostic process elements, it hence aligns with the business perspective of managing processes and data flows.

In particular, a data model inference mechanism was presented that enables a multi-level aggregation of partial data models into a global data schema. In addition, the combined model allows for real-time validation and consistency checks due to formal constraints on data type and cardinality consistency. This inference mechanism is essential for implementing model-driven techniques that require globally specified data models. MAML therefore achieves the desired balance of abstracting programming-heavy tasks to process flows of moderate complexity while keeping the technical expressiveness required for automatic source code generation.

Ongoing work focuses on an empirical evaluation to support the advantage of MAML over the related technical IFML notation, specifically with regard to its understandability by domain experts. Also, the MAML editor is further developed in order to feed back information from the inferred global data model into the modeling environment and provide the modeler with improved features for naming suggestions and real-time checks within the individual use case models.

Concerning a more general aspect that constitutes future work, applying the prototype to real-world problems might reveal further need for improvements. For example, data flow variations are currently limited to XOR elements due to the sequential proceeding on smartphone displays but might be extended if requested by practitioners. Furthermore, the platform-agnostic principle of MAML allows for its application to mobile devices beyond smartphones and tablets which opens up new possibilities for integrating business apps in everyday work practices. Regarding the emergence of novel devices such as smart watches, interesting questions arise concerning best practices for modeling and implementing apps on such devices with different input/output capabilities and user interaction patterns. Finally, it might be investigated to which extent previously existing process documentation can be reused to simplify app creation. For example, model to model transformations from/to other process modeling notations such as BPMN could be used to convert existing models into MAML use cases and just enrich them with missing pieces of information such as data objects.

References

[1] Roberto Acerbis, Aldo Bongio, Stefano Butti, and Marco Brambilla. Model-driven development of cross-platform mobile applications with webratio and ifml. In *Proceedings of the Second ACM International Conference on Mobile Software Engineering and Systems*, MOBILESoft '15, pages 170–171. IEEE Press, 2015.

[2] Apple Inc. Apple reinvents the phone with iphone. `http://www.apple.com/pr/library/2007/01/09Apple-Reinvents-the-Phone-with-iPhone.html`, 2007.

[3] Scott Barnett, Iman Avazpour, Rajesh Vasa, and John Grundy. A multi-view framework for generating mobile apps. *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 305–306, 2015.

[4] Bizness Apps. Mobile app maker — bizness apps. `http://biznessapps.com/`, 2016.

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

[6] Marco Brambilla, Andrea Mauri, and Eric Umuhoza. Extending the interaction flow modeling language (ifml) for model driven development of mobile applications front end. *Lecture Notes in Computer Science*, 8640:176–191, 2014.

[7] R. Breu, A. Kuntzmann-Combelles, and M. Felderer. New perspectives on software quality. *IEEE Software*, 31(1):32–38, 2014.

[8] Bubble Group. Bubble - visual programming. `https://bubble.is/`, 2016.

[9] Cinzia Cappiello, Maristella Matera, Matteo Picozzi, Alessandro Caio, and Mariano Tomas Guevara. Mobimash: End user development for mobile mashups. *WWW'12 - Proceedings of the 21st Annual Conference on World Wide Web Companion*, pages 473–474, 2012.

[10] I. Dalmasso, S. K. Datta, C. Bonnet, and N. Nikaein. Survey, comparison and evaluation of cross platform mobile application development tools. *2013 9th International Wireless Communications and Mobile Computing Conference, IWCMC 2013*, 2013.

[11] Jose Danado and Fabio Paternò. Puzzle: A visual-based environment for end user development in touch-based mobile phones. In Marco Winckler, Peter Forbrig, and Regina Bernhaupt, editors, *Human-Centered Software Engineering: 4th International Conference, HCSE 2012*, pages 199–216. Springer Berlin Heidelberg, 2012.

[12] Wafaa S. El-Kassas, Bassem A. Abdullah, Ahmed H. Yousef, and Ayman M. Wahba. Taxonomy of cross-platform mobile applications development approaches. *Ain Shams Engineering Journal*, pages –, 2015.

[13] Hidekazu Enjo and Junichi Iijima. Towards class diagram algebra for composing data models. In *Proceedings of the 2010 Conference on New Trends in Software Methodologies, Tools and Techniques: Proceedings of the 9th SoMeT_10*, pages 112–133, Amsterdam, The Netherlands, The Netherlands, 2010. IOS Press.

[14] Jan Ernsting, Christoph Rieger, Fabian Wrede, and Tim A. Majchrzak. Refining a reference architecture for model-driven business apps. *Proceedings of the 12th International Conference on Web Information Systems and Technologies (WEBIST 2016)*, pages 307–316, 2016.

[15] David Esperalta. Decsoft - App Builder. `https://www.davidesperalta.com/appbuilder`, 2016.

[16] R. B. France, S. Ghosh, T. Dinh-Trong, and A. Solberg. Model-driven development using uml 2.0: Promises and pitfalls. *Computer*, 39(2):59–66, 2006.

[17] Rita Francese, Michele Risi, Giuseppe Scanniello, and Genoveffa Tortora. Model-driven development for multi-platform mobile applications. In Pekka Abrahamsson, Luis Corral, Markku Oivo, and Barbara Russo, editors, *Product-Focused Software Process Improvement: 16th International Conference, PROFES 2015*, pages 61–67. Springer Intl. Publishing, 2015.

[18] Mirco Franzago, Henry Muccini, and Ivano Malavolta. Towards a collaborative framework for the design and development of data-intensive mobile applications. In *Proceedings of the 1st International Conference on Mobile Software Engineering and Systems*, MOBILESoft 2014, pages 58–61. ACM, 2014.

[19] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley professional computing series. Addison-Wesley, Reading, Mass., 1995.

[20] David Granada, Juan Manuel Vara, Marco Brambilla, Verónica Bollati, and Esperanza Marcos. Analysing the cognitive effectiveness of the webml visual notation. *Software & Systems Modeling*, 2015.

[21] H. Heitkötter and T. A. Majchrzak. Cross-platform development of business apps with md^2. In *Proc. of the 8th Int. Conf. on Design Science at the Intersection of Physical and Virtual Design (DESRIST)*, volume 7939 of *LNBIP*, pages 405–411. Springer, 2013.

[22] Zef Hemel and Eelco Visser. Declaratively programming the mobile web with mobl. In *Proceedings of the 2011 ACM International Conference on Object Oriented Programming Systems Languages and Applications*, OOPSLA '11, pages 695–712. ACM, 2011.

[23] International Organization for Standardization. Iso 5807:1985, 1985.

[24] Christopher Jones and Xiaoping Jia. Using a domain specific language for lightweight model-driven development. In *ENASE 2014, CCIS 551*, pages 46–62, 2015.

[25] David Knuplesch, Manfred Reichert, Linh Thao Ly, Akhil Kumar, and Stefanie Rinderle-Ma. Visual modeling of business process compliance rules with the support of multiple perspectives. In Wilfred Ng, Veda C. Storey, and Juan C. Trujillo, editors, *Conceptual Modeling: 32th International Conference, ER 2013*, pages 106–120. Springer Berlin Heidelberg, 2013.

[26] Yuehua Lin, Jeff Gray, and Frédéric Jouault. Dsmdiff: a differentiation tool for domain-specific models. *European Journal of Information Systems*, 16(4):349–361, 2007.

[27] Qichao Liu, Jeff Gray, Marjan Mernik, and Barrett R. Bryant. Application of metamodel inference with large-scale metamodels. *International Journal of Software and Informatics*, 6(2):201–231, 2012.

[28] Jesús J. López-Fernández, Jesús Sánchez Cuadrado, Esther Guerra, and Juan de Lara. Example-driven meta-model development. *Software & Systems Modeling*, 14(4):1323–1347, 2015.

[29] T. A. Majchrzak and J. Ernsting. Reengineering an approach to model-driven development of business apps. In *8th SIGSAND/PLAIS EuroSymposium 2015*, pages 15–31, 2015.

[30] T. A. Majchrzak, J. Ernsting, and H. Kuchen. Achieving business practicability of model-driven cross-platform apps. *OJIS*, 2(2):3–14, 2015.

[31] Marjan Mernik, Jan Heering, and Anthony M. Sloane. When and how to develop domain-specific languages. *ACM Comput. Surv.*, 37(4):316–344, 2005.

[32] Object Management Group. Business process model and notation, version 2.0, 2011.

[33] Object Management Group. Interaction flow modeling language, version 1.0, 2015.

[34] Object Management Group. Unified modeling language, version 2.5, 2015.

[35] Pentaho Corp. Data integration - kettle. http://pentaho.com/product/data-integration, 2016.

[36] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334–350, 2001.

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

[38] C. Simons and G. Wirtz. Modeling context in mobile distributed systems with the uml. *Journal of Visual Languages and Computing*, 18(4):420–439, 2007.

[39] Thomas Stahl and Markus Völter. *Model-Driven Software Development*. John Wiley & Sons, Chichester, 2006.

[40] E. Sutanta, R. Wardoyo, K. Mustofa, and E. Winarko. Survey: Models and prototypes of schema matching. *International Journal of Electrical and Computer Engineering*, 6(3):1011–1022, 2016.

[41] Eric Umuhoza and Marco Brambilla. Model driven development approaches for mobile applications: A survey. In Muhammad Younas, Irfan Awan, Natalia Kryvinska, Christine Strauss, and Do van Thanh, editors, *Mobile Web and Intelligent Information Systems: 13th International Conference, MobiWIS 2016*, pages 93–107. Springer Intl. Publishing, 2016.

[42] W.M.P. van der Aalst. Formalization and verification of event-driven process chains. *Information and Software Technology*, 41(10):639–650, 1999.

[43] D. Wolber. App inventor and real-world motivation. *SIGCSE'11 - Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, 2011.

[44] Uwe Zdun and M. Strembeck. Reusable architectural decisions for dsl design: Foundational decisions in dsl development. In *Proceedings of 14th European Conference on Pattern Languages of Programs (EuroPLoP 2009)*, pages 1–37, 2009.

[45] Kamil 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*, 43(1), 2015.

Working Papers, ERCIS

Nr. 1 Becker, J.; Backhaus, K.; Grob, H. L.; Hoeren, T.; Klein, S.; Kuchen, H.; Müller-Funk, U.; Thonemann, U. W.; Vossen, G.; European Research Center for Information Systems (ERCIS). Gründungsveranstaltung Münster, 12. Oktober 2004.

Nr. 2 Teubner, R. A.: The IT21 Checkup for IT Fitness: Experiences and Empirical Evidence from 4 Years of Evaluation Practice. 2005.

Nr. 3 Teubner, R. A.; Mocker, M.: Strategic Information Planning – Insights from an Action Research Project in the Financial Services Industry. 2005.

Nr. 4 Gottfried Vossen, Stephan Hagemann: From Version 1.0 to Version 2.0: A Brief History Of the Web. 2007.

Nr. 5 Hagemann, S.; Letz, C.; Vossen, G.: Web Service Discovery – Reality Check 2.0. 2007.

Nr. 6 Teubner, R.; Mocker, M.: A Literature Overview on Strategic Information Management. 2007.

Nr. 7 Ciechanowicz, P.; Poldner, M.; Kuchen, H.: The Münster Skeleton Library Muesli – A Comprehensive Overview. 2009.

Nr. 8 Hagemann, S.; Vossen, G.: Web-Wide Application Customization: The Case of Mashups. 2010.

Nr. 9 Majchrzak, T.; Jakubiec, A.; Lablans, M.; Ükert, F.: Evaluating Mobile Ambient Assisted Living Devices and Web 2.0 Technology for a Better Social Integration. 2010.

Nr. 10 Majchrzak, T.; Kuchen, H: Muggl: The Muenster Generator of Glass-box Test Cases. 2011.

Nr. 11 Becker, J.; Beverungen, D.; Delfmann, P.; Räckers, M.: Network e-Volution. 2011.

Nr. 12 Teubner, A.; Pellengahr, A.; Mocker, M.: The IT Strategy Divide: Professional Practice and Academic Debate. 2012.

Nr. 13 Niehaves, B.; Köffer, S.; Ortbach, K.; Katschewitz, S.: Towards an IT consumerization theory: A theory and practice review. 2012

Nr. 14 Stahl, F., Schomm, F., Vossen, G.: Marketplaces for Data: An initial Survey. 2012.

Nr. 15 Becker, J.; Matzner, M. (Eds.).: Promoting Business Process Management Excellence in Russia. 2012.

Nr. 16 Teubner, R.; Pellengahr, A.: State of and Perspectives for IS Strategy Research. 2013.

Nr. 18 Stahl, F.; Schomm, F.; Vossen, G.: The Data Marketplace Survey Revisited. 2014.

Nr. 19 Dillon, S.; Vossen, G.: SaaS Cloud Computing in Small and Medium Enterprises: A Comparison between Germany and New Zealand. 2015.

Nr. 20 Stahl, F.; Godde, A.; Hagedorn, B.; Köpcke, B.; Rehberger, M.; Vossen, G.: Implementing the WiPo Architecture. 2014.

Nr. 21 Pflanzl, N.; Bergener, K.; Stein, A.; Vossen, G.: Information Systems Freshmen Teaching: Case Experience from Day One (Pre-Version of the publication in the International Journal of Information and Operations Management Education (IJIOME)). 2014.

Nr. 22 Teubner, A.; Diederich, S.: Managerial Challenges in IT Programmes: Evidence from Multiple Case Study Research. 2015.

Nr. 23 Vomfell, L.; Stahl, F.; Schomm, F.; Vossen, G.: A Classification Framework for Data Marketplaces. 2015.

Nr. 24 Stahl, F.; Schomm, F.; Vomfell, L.; Vossen, G.: Marketplaces for Digital Data: Quo Vadis?. 2015.

Nr. 25 Caballero, R.; von Hof, V.; Montenegro, M.; Kuchen, H.: A Program Transformation for Converting Java Assertions into Controlflow Statements. 2016.

Nr. 26 Foegen, K.; von Hof, V.; Kuchen, H.: Attributed Grammars for Detecting Spring Configuration Errors. 2015.

Nr. 27 Lehmann, D.; Fekete, D.; Vossen, G.: Technology Selection for Big Data and Analytical Applications. 2016.

Nr. 28 Trautmann, H.; Vossen, G.; Homann, L.; Carnein, M.; Kraume, K.: Challenges of Data Management and Analytics in Omni-Channel CRM. 2017.