

Reengineering an Approach to Model-Driven Development of Business Apps

Tim A. Majchrzak¹(✉) and Jan Ernsting²

¹ ERCIS, University of Agder, Kristiansand, Norway
tima@ercis.de

² ERCIS, University of Münster, Münster, Germany
jan.ernsting@ercis.de

Abstract. Despite a perceived convergence in mobile application development, platforms such as Android and iOS remain largely incompatible. Supporting multiple platforms currently requires either separate native development (for each system) or utilization of a cross-platform development framework. While many such frameworks have been developed, only few are mature and even less are used widely, let alone commercially. Moreover, they typically are limited with regard to performance and to preserving a native look & feel. Worst of all, their usefulness for business apps is limited due to their low level of abstraction. In this paper, we take a closer look at an academic prototype that employs model-driven software development (MDSD) for a cross-platform framework that facilitates business app development. We discuss lessons learned from its development and early application, reengineering it with business producibility in mind. We aim at closing a design-oriented research gap: we describe what the approach to employ MDSD in mobile computing is and to what extent it might be useful in general. These findings are embedded in a case-study inspired discussion of the aims of reengineering the approach.

Keywords: MDSD · App · Mobile computing · Model-driven

1 Introduction

Within a few years, mobile devices such as smartphones and tables have become commonplace (cf. [1]). Even technology forecasts from the mid-2000s (e.g. by [65]) have been outpaced by rapid progress. Mobile devices are also increasingly used for business purposes [49]. Their platforms remain largely incompatible, though [45]. Software development kits (SDK), programming languages, frameworks, design guidelines and development standards for platforms such as Android, iOS and Windows Phone differ greatly [55]. However, these platforms enable applications (*apps*) to use a device to its full capabilities and – from a business viewpoint – to facilitate the adoption of mobile computing for business process improvements.

Supporting multiple platforms currently requires either separate native development or utilization of a cross-platform development framework. Strictly speaking, also Webapps can be used. HTML5 [7] is rather mature [18,31] yet limited compared to cross-platform development frameworks [29,32]. Thus, Webapps are not necessarily considered when comparing cross-platform approaches [52].

In case of native development, design and programming effort increases almost linearly with the number of supported platforms [34,55]. In case of using a cross-platform approach, the actual choice is quite limited. Despite a multitude of different possibilities, only so called hybrid approaches are widely used [32] – in fact, PhoneGap [8] a.k.a. Apache Cordova [4] arguably is the most widely used framework. Hybrid approaches are based on Web technology and enable rapid development for multiple devices [32]. They are limited with regard to performance and do not offer a native look & feel, though. A user interface that represents or at least resembles a native one has been described as very important for the use in *business apps* [47]. Business apps characterize a subset of applications for mobile devices; they typically are form-based and data-driven [34].

Cross-platform development approaches and the native SDKs alike pose an additional flaw when it comes to business app development: a low-level of abstraction. In case of native development, programming languages such as Java and Objective-C are used to tell the app programmatically *how* a problem is solved. Hybrid frameworks employ Web technologies such as HTML5 and JavaScript. This stills means that the focus is on *how* an app works. However, business apps typically are rather simple in functionality. They support or enable a business process but usually rely on (priorly existing) backend systems, which they use e.g. for computation [47]. Consequently, the desired way of developing a business app utilizes a high level of abstraction, i.e. you instruct an app *what* to do but let the framework decide *how* it is done. The latter can be different on each target platform. Hence, this amenity applies to both cross-platform work and for a single platform: refraining from programming but enabling app development for domain experts is *always* desirable.

We have closely observed the development of a scientific prototype for model-driven app development. MD² uses a domain-specific language (DSL) of its own and has been developed to facilitate the creation of business apps [33,34]. Since this approach is new to mobile computing, we revisit the design of MD². The research question for our work is: how can model-driven software development (MDS) be used for IS research? With this intentionally broad question we transport – and hopefully motivate for IS activities – a thread of research that has been discussed for software development for at least 15 years [40].

We will answer our research question with a focus on mobile computing. As steps towards this objective, we describe the background of MD²'s development (Sect. 2) and sketch the design method used (Sect. 3). Moreover, we present novel ideas that we deem feasible for reengineering the approach with business producibility in mind (Sect. 4). Based on this, we discuss our findings and generalize them (Sect. 5). Finally, we draw a conclusion (Sect. 6).

This paper makes several contributions. Firstly, we summarize MD² taking a third-party look and abstracting from the technical papers that describe its implementation. Secondly, we propose a design method for employing MDSD in IS research. Thirdly, we present reengineering ideas that underline the method. Fourthly, we highlight generalizable findings to give an outlook that should prove useful beyond our own work.

2 Background

Rightfully, the majority of development approaches describe *how* an app works i.e. in which – ideally innovative – way it solves a given problem. Yet, when engaging with different platforms stakeholders are faced with various approaches that vary not only in syntax but in underlying paradigms including aspects such as memory management and user interface components (*widgets*).

MD² provides a DSL that organises its element following the widely-used Model-View-Controller (MVC) design pattern [25]. By means of the MD² DSL, developers express *what* their app should resemble. MD² models describe relevant entities (*model*), their display (*view*), and behaviour. This is done with regard to both display induced manipulations, e.g. user triggered actions to persist an entity, and data dependent checks, e.g. checking an input’s conformance and ensuring that in depth checks are required for certain tasks (*controller*).

MD² DSL is neither specific to any platform nor does it rely on Web technology: it resembles a generative approach. For that generative purpose, the language is composed using Xtext [10]. It allows defining arbitrary languages in a syntax similar to that of the Extended Backus Naur Form (EBNF) [67]. Xtext instances are automatically converted into Eclipse Modeling Framework [61] models (a.k.a. *EMF models*). Typically, these models are then transformed to target platform code. This transformation draws from Xtend [9], a dialect for Java that among other features provides template expressions and dispatch methods that seamlessly blend in with Xtext and Java.

Other than Web-based approaches, MD² is not constrained to Web elements in terms of enhancing their namespace. Instead, MD² DSL is tailored to describe business apps. This is highlighted by its focus on data; in fact, the elements it provides are data-driven. For example, views may be generated from a given entity as shown in Listing 1. When an entity definition is combined with an auto generating view element, a view such as the one in Fig. 1 is created; here it is exemplarily shown for Android, but an according iOS view is created as well. Alternatively, each field of the resulting view can be defined manually.

```

1 package crm.contacts.models
2
3 entity CONTACT {
4     firstname : string
5     surname : string
6     phone : integer (optional)
7     email : string (optional)

```

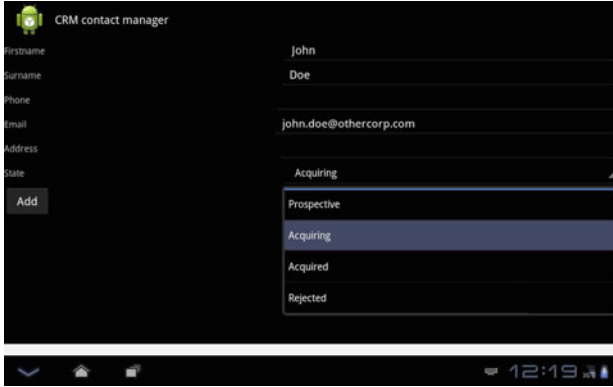


Fig. 1. View of a MD² generated app

```

8  address : string (optional)
9  state : ACQUISITIONSTATE
10 }
11
12 enum ACQUISITIONSTATE {
13   "Prospective", "Acquiring",
14   "Acquired", "Rejected"
15 }

```

Listing 1. MD² entity definition.

MD² adheres to the *convention of configuration* paradigm: while it is possible to adjust certain aspects, *sane* configurations are provided and enacted by default. This relieves modellers from dealing with tedious configurations whilst they could focus on the central aspects of their model. For example, the auto generating view element can be told to use only selected attributes or exclude certain attributes from the resulting view. Thus, little model code is required to produce apps. Though lines of code may have limited explanatory value, MD²'s lightweight models produce apps with useful features per default. These include device local persistence, network communications for remote interactions and persistence that are based on a well-defined RESTful Web service interface [66].

While only one of us was involved in the development of MD², we were able to closely survey its development process. The development of the MD² generators is based on the reference implementation proposed by [60]. Therefore, apps for the selected platforms (iOS and Android) were developed from scratch. These apps featured common elements that are required in business apps. On that basis, commonalities of the platforms were identified and problems due to different approaches on the platforms were solved. Most of all, the paradigm mismatch of the Objective-C and Java programming languages as well as the platform conventions caused difficulties in the identification process. Nonetheless, feasible abstractions for all platforms were identified and are represented in MD² DSL.

For a more elaborate discussion of MD² with a focus on technological details, please refer to the respective papers from the designers and developers [33–35].

3 Methodology

This section serves two purposes. Firstly, it explains the approach to writing this paper. Secondly, it sketches the methodology behind developing MD² with the goal of giving generalizable advice. We deem this of particular relevance since the software engineering research [48] presented with MD² follows a method typical for that kind of research without explaining too much of it.

The foundation for our aim of contributing to the theory of mode-driven software development in IS research is the prototype described in the prior section. It is not in a commercial-ready state yet sophisticated enough to leave the technological perspective of its authors. This scrutinisation of an MDSD project allows to consider its development process. To avoid being stuck in prior work, we amend our paper with a discussion of improvements. The idea is to present another main iteration whereas developing MD² has been incremental and iterative already. This serves a methodological aim: while from the technological perspective small iterations lead to the fulfilment of functional requirements, large iterations – i.e. reengineering – facilitates the fulfilment of non-functional requirements. This in particular concerns business aims.

The leading method behind MD² is the common approach followed in prototype development in computer science (CS). It closely aligns with *design science* as the arguably most profound methodology for design-oriented artefact construction in IS research [38]. In fact, software engineering as the particular CS discipline is specifically suited for integration with design science [50]. Applying a sound methodological approach to a problem of profound industry relevance is a precondition to be both rigorous and relevant. After the initial, very rough idea of applying MDSD to cross-platform development was discussed, development of MD² was carried out iteratively. It thereby followed the *design science cycle* of building and evaluating a prototype as described in [39].¹

The realization of MD² roughly followed the *design science research process* [54]. We retroactively apply the steps to MD² in a similar fashion. Part of the information we use here can be derived from the published papers on MD² but we also draw from our contact to the developers.

Firstly, a problem was identified and motivated. When work on MD² was initiated, the available frameworks for cross-platform development were much less mature than today and the demand for cross-platform solutions was rising [36]. As a particular motivation, an approach was sought that would be suitable to businesses by refraining from low-level programming and by offering the possibility to align with business processes (for early thoughts on MDSD business process aligning cf. with the work by [57]). MDSD appeared to be a natural choice here; at the same time its feasibility for mobile computing had hardly

¹ From the viewpoint of the natural sciences, the approach might also be called *engineering science* as rather aggressively proposed by Gruner and Groeze [27].

been assessed scientifically and not exhaustively tried out by practitioners (cf. also with the discussion of related work in [34]).

Secondly, objectives for a solution were defined. In case of MD², these were the choice of MDSD and the limitation to business apps, which were a compromise of the approaches' general value and technological obstacles that would need to be overcome. While most objectives were qualitative, also quantitative objectives could be identified, such as a faster time-to-market for apps [55] and a reduction in lines of code to be written, both in comparison to native development. The objectives were considered for the design and development, which is the third activity that is proposed in [54]. Strictly speaking, the result was not a single artefact but several artefacts such as the generators and other components, which have value beside their role as part of MD².

Fourthly, MD² was demonstrated in several ways. Following internal experiments by the development team, a case study was implemented in cooperation with a business partner using an actual case from one of their customers – a tariff calculator [34]. As the next step, the technological progress was presented in a number of papers that addressed both computer science [34,35,37] and information systems [33] audiences.

As a consequence, the fourth activity overlapped with the fifth and the sixth activity proposed in [54]: *evaluation* and *communication*, respectively. Experiments also served as a first benchmark. The successive implementation of the tariff calculator led to further insights. Finally, presentation at conferences stimulated discussion and at the same time allowed disseminating the findings.

Finishing the first experiments, a continuous cycle of refinements was started (repeating activities three through six). Observed implications lead to smaller refinements [38, p.5], particularly concerning the user interface of generated apps and the generators. However, it did not comprise of changes to the domain specific language save small amendments. This was a deliberate choice: changing the DSL would render most (probably all) prior applications of it useless.² In contradiction to application programming interfaces, where new methods might be implemented while marking the old ones as **deprecated**, changing the DSL typically is not downward compatible.

We propose that the consequence for MD² – as well as for other uses of MDSD in IS – is to have a design science process *nested* in a design science process. We have illustrated this idea in Fig. 2. There is an *outer* cycle of design and evaluate whereas its design phase comprises of an *inner* cycle of design and evaluate itself as well as a reengineering phase. In our view, MD² should now finish its first round in the inner cycle and enter the reengineering phase, allowing fundamental changes based on the gained insights. It then needs to undergo an exhaustive evaluation that initiates the second inner circle of small design changes and detailed evaluation with several methods and in several settings.

² Apps implemented in the then old version of the DSL would be incompatible with the new generators, as would apps implemented with the new version of the DSL be with the old generators.

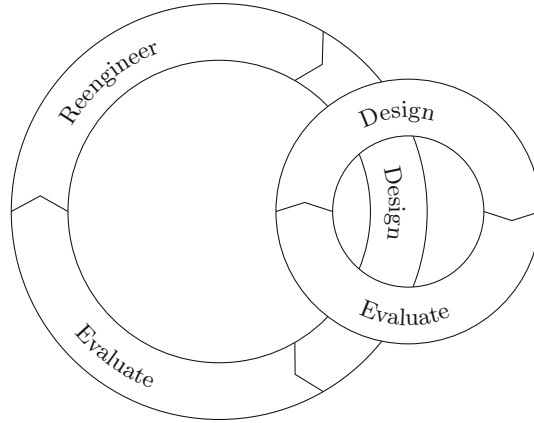


Fig. 2. Nested design science cycle

It might be argued that it is inherent to design science research to have smaller and larger iterations. For example, the cross-platform framework applause [14, p. 126] has explicitly been developed in an iterative process. It might even be argued that this approach is routinely followed in prototype development in computer science, even though evaluation is typically conducted as experiments. Nevertheless, we deem the idea of a fundamental cycle and a nested small-steps cycle to be particular to MDSD projects. In fact, there is not simply an *alternating* process of small and large changes. The small steps eventually facilitate a *milestone*, which is finalized when ending a cycle. While reengineering is a design activity as well, it is different in scope and in aims. Even with taking account all lessons learned from the prior activities, reengineering puts the stakes much higher. By yielding a new prototype that is neither downwards nor upwards compatible to the former version, this design activity is a *leap* compared to the steps undertaken in the inner circle. Moreover, while steps can be well anticipated, ideally avoiding any ill choice, reengineering always poses the risk of introducing a fundamental flaw. This can mostly be avoided by arduous work, but nevertheless reengineering will always lead to new challenges that have to be overcome. This can be seen as a consequence of applying changes to the original idea.

To motivate an example for changes to the DSL: ideally, anything the resulting application is capable of should be expressible in the DSL. However, particularly when the platforms that native code is generated for differ greatly, finding a kind of lowest common denominator becomes cumbersome. In fact, it might hinder effective usage of the DSL. To overcome such problems, the *generation gap* pattern [24, pp. 571ff.] can be used. Simply speaking, it allows to specify gaps at which code can be freely inserted in the generated code. Typically, design of a DSL would start without considering *gaps*. If they become necessary – e.g. when extending the DSL but realizing that not all desired extensions can reasonably be implemented – reengineering rather than simply adding them is the natural choice due to the fundamental impact of the change.

At this point, an intermediary conclusion can be drawn. It also serves as the transition to the next section, which describes the first steps towards a cycle of reengineering. The process of developing MD² well aligns with design science research. As best practices were followed with regard to the employed tools, the construction of generators, and the design of the DSL, the process stands as an example of a MDSD project in mobile computing. The solved problem comes from the domain of IS research. Thus, we even deem the process to be exemplarily for MDSD in IS research. As a consequence, we propose the nested process with explicit reengineering for similar projects. While most parts of the projects would do well with the core design science process, DSL design demands explicit and deliberate reengineering rather than incremental design.

4 Reengineering

Besides discussing shortcomings in the MD² DSL, this section also goes beyond the scope of currently supported platforms and touches upon an advanced pre-processing to reduce the burden of implementing generators for new platforms as well as easing the maintenance for existing generators. As motivated earlier, the proposed changes are not mere steps but require reengineering.

MD² can be criticized for its approach of defining input elements in view declarations with regard to two aspects. First, text input definitions were not only used for text but also for numbers, dates, time, and timestamps (in conjunction with validators). Secondly, despite platforms such as iOS lacking these, input declarations for Boolean values are represented as so called *check boxes*. Originally, this naming convention stemmed from Android; yet, in the latest Android release, check boxes were replaced by switches [3]. This causes a divergence in meaning and introduces ambiguity where explicitness could be employed.

Given its data-driven nature, providing direct correspondences for each data type when declaring views – i.e. offering dedicated input elements for numbers, dates, time, and timestamps – constitutes a straightforward improvement to MD². In the first place, MD² required both the input declaration and an according validator to determine the input element that best suits the corresponding platform. Table 1 shows the reengineered input notation for non-textual inputs, which no longer require accompanying validators.

Furthermore, the reengineered notation enables the provision of improved editor validation as incompatible data types and input elements are highlighted. Instead of having to wait for a full transformation and deploy process to finish, users receive instant feedback concerning invalidly used input elements.

For utilization by businesses, i.e. users wanting to generate apps for their respective target platform(s), MD²'s generation step is a vital component to adjust. However, MD²'s developers consider the development and maintenance of generators a laborious and time consuming endeavour [22]. So far, generators perform the transformations from model to code. An intermediate preprocessing stage already helps to enrich the original model with model to model transformation. This reduces the overall work that generators perform. Preprocessing

Table 1. Comparison of the Original and the Reengineered Notation for Input Fields

Data type	Original notation	Reengineered notation
string	<code>TextInput ID</code>	<code>TextInput ID</code>
integer	<code>TextInput ID + IsIntegerValidator</code>	<code>IntegerInput ID</code>
float	<code>TextInput ID + IsNumberValidator</code>	<code>NumberInput ID</code>
boolean	<code>CheckBox ID</code>	<code>BooleanInput ID</code>
date	<code>TextInput ID {type date} + DateFormatValidator</code>	<code>DateInput ID</code>
time	<code>TextInput ID {type time} + DateFormatValidator</code>	<code>TimeInput ID</code>
datetime	<code>TextInput ID {type datetime} + DateFormatValidator</code>	<code>DatetimeInput ID</code>
Enum	<code>SelectBox ID</code>	<code>OptionInput ID</code>

allows to introduce novel language elements that necessitate any changes to the generators. For example, the introduction of auto generative views – which provide input fields for a given entity without further ado – is one case where the language elements for the auto generative views do not have to be considered by generators as they can be converted into well-defined views by preprocessing. Preprocessing also includes the inference of correct inputs to use (cf. the above discussion of input elements) as well as the transformation of sequences of views that could be nested into flattened sequences.

To further ease generator maintenance and development, minimising the number of elements generators have to support reduces overall effort. A subset of MD² language elements was identified for that purpose. This subset suffices to represent more complex language elements such as “workflows” and conditional event handling. To identify the *sweet spot*, all language elements require investigation to determine whether they could be represented through others.

We identified several core language elements that suffice to represent complex components such as view sequences, conditional events, combined actions, and validators. For example, validation of string lengths can be automatically transformed to reuse regular expression validators.

To demonstrate the effectiveness of the improved preprocessing, MD²’s language elements used for describing sequences of views were tuned to support advanced flows of views. Despite adding the capability for conditional branching, generators were not required to account for this addition at all. In fact, elements that describe view sequences were removed from the preprocessed model altogether, i.e. generators no longer need to take care of these elements. As a result of the improved preprocessing, generators have to support the core language elements only. The overall number of elements in a preprocessed model certainly exceeds that of its original model. This, however, does not pose a problem since the preprocessed model is intended for fully automatic generation. These core language elements not only represent a subset of the overall MD² DSL, but

can also be characterised and described in an unambiguous way, such that the generated apps exhibit consistent behaviour across different platforms.

Reengineering might go beyond the above sketched ideas. With Xbase, Efttinge et al. propose means to include behavioural aspects in DSLs [21]. The idea is to keep the high level of abstraction DSLs provide while not being forced to “fall back” to general purpose programming languages. With “language inheritance” from Xtend [21], Xbase could provide new means to extend MD²’s Java backend. That way, extending the backend’s capabilities would no longer require modifications to the generated backend code or utilising means such as the *generation gap* (cf. Sect. 3). Instead, adjustments are dealt with as first class model artefacts.

5 Discussion

The presented background and insights allow for a discussion. We first highlight related work. Then we apply our understanding of MDSD to IS research in general. This subsequently allows to name limitations and to draw an outlook.

5.1 Related Work

Related approaches have already been named throughout the paper. Nevertheless, some light should be shed on closely related work.

Behrens [14] motivates MDSD for creating iPhone apps using *applause*. Although its support was extended to Windows Phone and Android, it remains limited to displaying data. The development process appears to be comparable to that of MD² – it is not motivated from a design science perspective, though. AXIOM [41] takes a more technical way; it features aspects of UML and uses the programming language Groovy. Employing a graphic modelling interface, *Modagile* [2] follows a different approach altogether. However, both Modagile and AXIOM require manually performed adjustments such as adding code for control logic or creating mappings for the transformation process. Current progress has been reported for AXIOM [42], which is promising. Keeping track of the parallel development of AXIOM and MD² might yield further insights in the near future.

Literature on model-driven development of apps is vast; at the same time, almost all articles only loosely relate to our work for they neither focus on cross-platform development nor highlight the underlying process. Thereby, they can also be seen as additional motivation for our work.

Balagtas-Fernandez and Hussmann [11] proposed in 2008 that MDSD is feasible for mobile computing. Considering the year of publication, this is particularly notable for closely aligning with the breakthrough of smartphones. Their main concern, however, were graphical interfaces. Thus, their work can be attributed to the domain of human computer interaction (HCI). The same is true for the work by Diep, Tran and Tran [19]. Notable is their focus on cross-platform user interface generation.

In alignment with our ideas but with a variety of different focuses, other authors suggest a higher level of abstraction. This can be recorded for mobile architectures [20], testing [56], self-adaptivity [26, 58], and context-awareness [16, 30]. Some authors also propose specific applications. Examples are a work on healthcare apps [43] and the general approach of [53] for Android-based devices.

5.2 MDSD in Information Systems Research

MDSD is broadly applied in computer science research. Models are a key concept in information systems (IS) research. As a consequence, we propose to utilize MDSD much more in IS projects. In particular, we suggest that modelling in business terms – e.g. as part of business process modelling (BPM) [12] – and MDSD are integrated. For the case of MD², this could e.g. mean that workflows as modelled in enterprises can be incorporated as the workflow in apps.

The idea to use MDSD in IS research is not new. Castro and colleagues [17] propose an alignment with “high level business models” in the context of service-oriented information systems. Despite a different focus, this idea is very close to ours. Unfortunately, such threads of research do not seem to have been sustainably followed. Moreover, most work explicitly highlights its technical contribution (cf. [64]). This (i.e. the technical contribution) has much merit; in fact, this paper would not have been written without the technological progress described as a result from developing MD². However, it is highly desirable to contribute to theory besides technological progress. In fact, technological progress and advancements in IS methods can go hand-in-hand.

MDSD in information systems development has been particularly considered in the area of security [13, 23]. Whereas security is a topic within IS research, typical papers can rather be attributed to computer science and do not much focus on methodology; generalization consequently focuses on security, not MDSD.

MDSD offers a particularly good background for projects that facilitate “learning via making” [51, p. 111]. Moreover, it is adequate for coming up with solutions relevant for business, yet *satisficing* [59] from a scientific point of view. In other words: while there are good reasons just from a problem-solving point of view already, MDSD can iteratively be applied to gain new insights. Business interest in MDSD had risen [15],³ which helps to disseminate progress beyond the scientific literature. Moreover, it is very helpful to be able to gain *real* cases from enterprises.

MDSD research can draw from a profound theoretical base. At the same time, its application to so far undiscovered fields only needs to rely on this basis with regard to methodology or by means of analogies. It thereby aligns with a strength of design science research in general, i.e. to enable insights without necessarily relying on a broad theoretical base [63].

Tool support for MDSD is very advanced [44]. As explained for MD², with the availability of tools such as Xtext and Xtend many formerly laborious steps

³ Cases are particularly reported for model-driven architecture (MDA) [28], which can be used in combination with MDSD and underlines the interest.

are eased; much manual work can be avoided and MDSD prototypes can be created with a focus on high-level work. In our eyes this is yet another argument for using MDSD more widely.

The above arguments in combination with our proposal of a nested design science process (see Sect. 3) led us to a conclusion. We deem MDSD to be very appropriate for design-oriented research projects in IS research. Reflecting on the development of MD², there without question is room for improvement. Nevertheless, the general approach is sound.

5.3 Limitations

Limitations have to be considered from two perspectives. First, some limitations lie in the approach we took in this paper. Second, there are limitations that are inherent to MDSD for IS research.

MD² is a research prototype. As a direct consequence, it inherently has a work-in-progress character although papers present milestones, which, by their contribution, mark finished work. The main limitation of MD² as the case for our paper thus is the fact that business producibility has not yet been reached.

There are two limitations this paper poses. Firstly, it relies on one main case and is not quantitatively verified. However, since we present novel ideas, which should be discussed in detail, we deem a quantitative assessment to be way too early. Secondly, this paper combines two contributions, one concerning method and one concerning a prototype's reengineering. An elaborate paper on the reengineering alone would have made a sound contribution – but for a computer science audience. A method paper without motivation and case would have been purely theoretical. Therefore, we have combined method and innovative case. This deliberate choice is a strength of this paper but a weakness at the same time due to the presentation of more than one message at a time.

Identifying the general limitations of MDSD in IS research is not straightforward. In fact, the lack of work on this topic is a limitation by itself. Drawing the border of MDSD for IS would be merely speculative at this point. Therefore, we will tackle this topic as future work. However, neither do the limitations of our research hinder its merits nor are the boundaries of MDSD “show stoppers” in terms of the applicability to IS.

5.4 Outlook and Future Work

As a consequence from our findings, and based on the presented discussion, an outlook has to be drawn. Much future work remains to be done.

With regard to cross-platform development approaches based on MDSD, many iterations of designing and iterating will be required before business producible tools will be available. Moreover, extended qualitative assessment followed by quantitative studies is required.

While we are convinced of the feasibility of MDSD in app development, it remains to be seen what future approaches will look like. In fact, in the few years

of widespread usage of smartphones and tablets, the landscape of development methods, tools, and frameworks has changed dramatically. Whether platforms will converge so much that cross-platform development becomes unnecessary, HTML5-based development or a Hybrid approach such as PhoneGap win the race, or a completely different approach – such as MDSD – taking the lead is impossible to anticipate. However, the dynamic of change makes research in this field not only challenging but also exciting.

As a side note, it will also be important to monitor other paradigms. We have argued that Webapps are appropriate for cross-platform development in some cases, but problematic in others. Walker, Turnbull, and Sim suggest that users prefer browsing “normal” Web pages to those that have mobile versions [65]. It would be interesting to check whether this also applies to using Web technology as part of cross-platform frameworks that do not use native elements.

There are other approaches that should be closely monitored for further development. One example is Google Inbox [62]. Based on a Java data model that is used natively on Android, compilation to JavaScript is performed using the Google Widget Toolkit [6]. Then cross-compiling to Objective-C using J2ObjC [5] is conducted. Their conclusion is also true for an MDSD approach. To become an option, apps need “significant UI independent client logic”, support multiple platforms, and “must not compromise on user experience and polish” [62].

Besides this general look at mobile computing, we are looking forward to seeing more IS projects that employ MDSD. Whether we and others can actually stimulate an increased consideration remains to be seen.

Our own work will continue in several threads of research. Firstly, we will contribute to the development of MD², aiming at a new version based on the reengineering. We also intend to work with partners from industry on its evaluation, ideally (and eventually) achieving a version that can be released for first commercial projects. Secondly, we will continue work on the IS and business perspective on cross-platform app development [46]. We seek to foster the understanding of business apps and of the integration of business processes with app development and deployment. Thirdly, we plan to intensify our contribution to the research of methodology for we consider MDSD to be underrepresented in IS research despite IS’s strong focus on models and modelling.

6 Conclusion

In this paper we presented the background of MD², an academic prototype for the model-driven development of cross-platform apps. We highlighted insights concerning the development process and propose its suitability for MDSD project in information systems research. Moreover, we discussed the reengineering of MD², thereby demonstrating the process and contributing to the theory on MDSD in mobile computing. We amended our paper with arguments and thoughts concerning the status quo of MDSD in IS. While we seek to discuss our ideas, several threads of future work could also be identified.

Acknowledgements. We would like to thank Sören Evers and con terra GmbH for their contribution to the reengineering of MD². Sören’s work is particularly reflected in Table 1. Additionally, we would like to thank Klaus Fleerkötter, Daniel Kemper, Sandro Mesterheide and Jannis Strodtkötter for their contribution to the first version of MD².

References

1. Gartner Press Release, February 2013. <http://gartner.com/newsroom/id/2665715>
2. Modagile Mobile (2013). <http://www.modagile-mobile.de/>
3. Android Styleguide (2014). <https://developer.android.com/design/building-blocks/switches.html>
4. Apache Cordova (2014). <http://cordova.apache.org/>
5. google/j2objc (2014). <https://github.com/google/j2objc>
6. GWT (2014). <http://www.gwtproject.org/>
7. HTML5 (2014). <http://www.w3.org/TR/html5/>
8. PhoneGap (2014). <http://phonegap.com/>
9. Xtend (2014). <http://www.eclipse.org/xtend/>
10. Xtext (2014). <http://www.eclipse.org/Xtext/>
11. Balagtas-Fernandez, F.T., Hussmann, H.: Model-driven development of mobile applications. In: Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering, ASE 2008, pp. 509–512. IEEE Computer Society, Washington, DC (2008)
12. Barjis, J.: The importance of business process modeling in software systems design. *Sci. Comput. Program.* **71**(1), 73–87 (2008)
13. Basin, D., Clavel, M., Egea, M.: A decade of model-driven security. In: Proceedings of the 16th ACM Symposium on Access Control Models and Technologies, SACMAT 2011, pp. 1–10. ACM, New York (2011)
14. Behrens, H.: MDSD for the iPhone: developing a domain-specific language and ide tooling to produce real world applications for mobile devices. In: Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion, pp. 123–128. ACM, New York (2010)
15. Brambilla, M., Cabot, J., Wimmer, M.: Model Driven Software Engineering in Practice. Morgan & Claypool, USA (2012)
16. Carton, A., Clarke, S., Senart, A., Cahill, V.: Aspect-oriented model-driven development for mobile context-aware computing. In: Proceedings of the 29th International Conference on Software Engineering Workshops, ICSEW 2007, pp. 191–198. IEEE Computer Society, Washington, DC (2007)
17. Castro, V.d., Mesa, J.M.V., Herrmann, E., Marcos, E.: A model driven approach for the alignment of business and information systems models. In: Proceedings of the 2008 Mexican International Conference on Computer Science, ENC 2008, pp. 33–43. IEEE Computer Society, Washington, DC (2008)
18. Curran, K., Bond, A., Fisher, G.: HTML5 and the mobile web. *Int. J. Innov. Digit. Econ. (IJIDE)* **3**(2), 40–56 (2012)
19. Diep, C.K., Tran, Q.N., Tran, M.T.: Online model-driven ide to design guis for cross-platform mobile applications. In: Proceedings of the Fourth Symposium on Information and Communication Technology, SoICT 2013, pp. 294–300. ACM, New York (2013)

20. Dunkel, J., Bruns, R.: Model-driven architecture for mobile applications. In: Abramowicz, W. (ed.) BIS 2007. LNCS, vol. 4439, pp. 464–477. Springer, Heidelberg (2007)
21. Efftinge, S., Eysholdt, M., Köhnlein, J., Zarnekow, S., von Massow, R., Hasselbring, W., Hanus, M.: Xbase: implementing domain-specific languages for Java. *SIGPLAN Not.* **48**(3), 112–121 (2012)
22. Evers, S., Fleerkötter, K., Kemper, D., Mesterheide, S., Strodtkötter, J.: MD² model-driven mobile development (2012). <http://www-pi.github.io/md2-web/res/MD2-Documentation.pdf>
23. Fernández-Medina, E., Jurjens, J., Trujillo, J., Jajodia, S.: Editorial: model-driven development for secure information systems. *Inf. Softw. Technol.* **51**(5), 809–814 (2009)
24. Fowler, M.: Domain-Specific Languages. Addison-Wesley Pearson Education, Upper Saddle River (2011)
25. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Boston (1995)
26. Geihs, K., Reichle, R., Khan, M.U., Solberg, A., Hallsteinsen, S.: Model-driven development of self-adaptive applications for mobile devices: (research summary). In: Proceedings of the 2006 International Workshop on Self-adaptation and Self-managing Systems, SEAMS 2006, pp. 95–95. ACM, New York (2006)
27. Gruner, S., Kroeze, J.: On the shortage of engineering in recent information systems research. In: Proceedings of the 25th Australasian Conference on Information Systems. ACIS (2014)
28. Guttman, M., Parodi, J.: Real-Life MDA: Solving Business Problems with Model Driven Architecture. Morgan Kaufmann Publishers Inc., San Francisco (2007)
29. Hallem, S.: overcoming html5's limitations (2013). <http://drdobbs.com/web-development/overcoming-html5s-limitations/240159696>
30. Harchay, A., Cheniti-Belcadhi, L., Braham, R.: A model driven infrastructure for context-awareness mobile assessment personalization. In: Proceedings of the 2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications, TRUSTCOM 2012, pp. 1676–1683. IEEE Computer Society, Washington, DC (2012)
31. Harjono, J., Ng, G., Kong, D., Lo, J.: Building smarter web applications with HTML5. In: Proceedings of the 2010 Conference of the Center for Advanced Studies on Collaborative Research, pp. 402–403. ACM, New York (2010)
32. Heitkötter, H., Hanschke, S., Majchrzak, T.A.: Evaluating cross-platform development approaches for mobile applications. In: Cordeiro, J., Krempels, K.-H. (eds.) WEBIST 2012. LNBIP, vol. 140, pp. 120–138. Springer, Heidelberg (2013)
33. Heitkötter, H., Majchrzak, T.A.: Cross-platform development of business apps with MD². In: vom Brocke, J., Hekkala, R., Ram, S., Rossi, M. (eds.) DESRIST 2013. LNCS, vol. 7939, pp. 405–411. Springer, Heidelberg (2013)
34. Heitkötter, H., Majchrzak, T.A., Kuchen, H.: Cross-platform model-driven development of mobile applications with MD². In: Proceedings SAC 2013, pp. 526–533. ACM (2013)
35. Heitkötter, H., Majchrzak, T.A., Kuchen, H.: MD2-DSL - eine domänenspezifische Sprache zur Beschreibung und Generierung mobiler Anwendungen. In: Wagner, S., Lichter, H. (eds.) ATPS 2013. LNI, vol. 215, pp. 91–106. GI, Bonn (2013)
36. Heitkötter, H., Majchrzak, T.A., Wolfgang, U., Kuchen, H.: Business Apps: Grundlagen und Status quo. No. 4 in Working Papers, Förderkreis der Angewandten Informatik an der WWU Münster e.V. (2012)

37. Heitkötter, H., Kuchen, H., Majchrzak, T.A.: Extending a model-driven cross-platform development approach for business apps. *Sci. Comput. Program.* **97**, Part 1(0), 31–36 (2015)
38. Hevner, A.R., Chatterjee, S.: *Design Research in Information Systems: Theory and Practice*. Springer, Heidelberg (2010)
39. Hevner, A.R., March, S.T., Park, J., Ram, S.: Design science in information systems research. *MIS Q.* **28**(1), 75–105 (2004)
40. Humm, B., Schreier, U., Siedersleben, J.: Model-driven development — hot spots in business information systems. In: Hartman, A., Kreische, D. (eds.) *Model Driven Architecture – Foundations and Applications*. LNCS, vol. 3748, pp. 103–114. Springer, Heidelberg (2005)
41. Jia, X., Jones, C.: AXIOM: a model-driven approach to cross-platform application development. In: *ICSOFT 2012 - Proceedings of the 7th International Conference on Software Paradigm Trends*, pp. 24–33 (2012)
42. Jones, C., Jia, X.: The AXIOM model framework - transforming requirements to native code for cross-platform mobile applications. In: Filipe, J., Maciaszek, L.A. (eds.) *ENASE 2014 - Proceedings of the 9th International Conference on Evaluation of Novel Approaches to Software Engineering*, pp. 26–37. SciTePress (2014)
43. Khambati, A., Grundy, J., Warren, J., Hosking, J.: Model-driven development of mobile personal health care applications. In: *Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering, ASE 2008*, pp. 467–470. IEEE Computer Society, Washington, DC (2008)
44. Kurtev, I., Bézin, J., Jouault, F., Valduriez, P.: Model-based DSL frameworks. In: *Companion to the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications*, pp. 602–616. ACM, New York (2006)
45. Lin, F., Ye, W.: Operating system battle in the ecosystem of smartphone industry. In: *Proceedings 2009 International Symposium on Information Engineering and Electronic Commerce (IEEC)*, pp. 617–621. IEEE CS (2009)
46. Majchrzak, T.A., Ernsting, J., Kuchen, H.: Achieving business practicability of model-driven cross-platform apps. *Open J. Inf. Syst. (OJIS)* **2**(2), 3–14 (2015)
47. Majchrzak, T.A., Heitkötter, H.: Status quo and best practices of app development in regional companies. In: Krempels, K.-H., Stocker, A. (eds.) *WEBIST 2013*. LNBIP, vol. 189, pp. 189–206. Springer, Heidelberg (2014)
48. Marcos, E.: Software engineering research versus software development. *SIGSOFT Softw. Eng. Notes* **30**(4), 1–7 (2005)
49. McLellan, C.: enterprise mobility in 2014: App-ocalypse now? (2014). <http://www.zdnet.com/enterprise-mobility-in-2014-app-ocalypse-now-7000028499/>
50. Morrison, J., George, J.: Exploring the software engineering component in MIS research. *Commun. ACM* **38**(7), 80–91 (1995)
51. Oates, B.J.: *Researching Information Systems and Computing*. Sage Publications, London (2005)
52. Ohrt, J., Turau, V.: Cross-platform development tools for smartphone applications. *IEEE Comput.* **45**(9), 72–79 (2012)
53. Parada, A.G., Brisolara, L.B.D.: A model driven approach for android applications development. In: *Proceedings of the 2012 Brazilian Symposium on Computing System Engineering, SBESC 2012*, pp. 192–197. IEEE Computer Society, Washington, DC (2012)

54. Peffers, K., Tuunanen, T., Rothenberger, M., Chatterjee, S.: A design science research methodology for information systems research. *J. Manag. Inf. Syst.* **24**(3), 45–77 (2007)
55. Ribeiro, A., da Silva, A.R.: Survey on cross-platforms and languages for mobile apps. In: *Proceedings of the 2012 Eighth International Conference on the Quality of Information and Communications Technology, QUATIC 2012*, pp. 255–260. IEEE Computer Society, Washington, DC (2012)
56. Ridene, Y., Barbier, F.: A model-driven approach for automating mobile applications testing. In: *Proceedings of the 5th European Conference on Software Architecture: Companion Volume, ECSA 2011*, pp. 9:1–9:7. ACM, New York (2011)
57. Ruokonen, A., Pajunen, L., Systa, T.: On model-driven development of mobile business processes. In: *Proceedings of the 2008 Sixth International Conference on Software Engineering Research, Management and Applications, SERA 2008*, pp. 59–66. IEEE Computer Society, Washington, DC (2008)
58. Schmidt, H., Dang, C.T., Gessler, S., Hauck, F.J.: Model-driven development of adaptive applications with self-adaptive mobile processes. In: Meersman, R., Dillon, T., Herrero, P. (eds.) *OTM 2009, Part I. LNCS*, vol. 5870, pp. 726–743. Springer, Heidelberg (2009)
59. Simon, H.A.: *The Sciences of the Artificial*, 3rd edn. MIT Press, Cambridge (1996)
60. Stahl, T., Völter, M.: *Model-Driven Software Development*. Wiley, New York (2006)
61. Steinberg, D., Budinsky, F., Paternostro, M., Merks, E.: *EMF: Eclipse Modeling Framework*, 2nd edn. Addison-Wesley, Boston (2009)
62. Toubassi, G.: Going under the hood of inbox (2014). <http://gmailblog.blogspot.de/2014/11/going-under-hood-of-inbox.html>
63. Vaishnavi, V.K., Kuechler Jr, W.: *Design Science Research Methods and Patterns: Innovating Information and Communication Technology*. Auerbach Publications, Boston (2007)
64. Vara, J.M., Marcos, E.: A framework for model-driven development of information systems: technical decisions and lessons learned. *J. Syst. Softw.* **85**(10), 2368–2384 (2012)
65. Walker, M., Turnbull, R., Sim, N.: Future mobile devices: an overview of emerging device trends, and the impact on future converged services. *BT Technol. J.* **25**(2), 120–125 (2007)
66. Webber, J., Parastatidis, S., Robinson, I.: *REST in Practice. Hypermedia and Systems Architecture*. O'Reilly, Cambridge (2010)
67. Wirth, N.: What can we do about the unnecessary diversity of notation for syntactic definitions? *Commun. ACM* **20**(11), 822–823 (1977)

Information Systems: Development, Applications,
Education

8th SIGSAND/PLAIS EuroSymposium 2015, Gdansk,
Poland, September 25, 2015, Proceedings

Wrycza, S. (Ed.)

2015, X, 171 p. 28 illus. in color., Softcover

ISBN: 978-3-319-24365-8