A Method for Ontology-Driven Minimum Viable Platform Development

Thomas Derave[1] (Corresponding author) [0000-0003-1547-8333], Tiago Prince Sales[2][0000-0002-5385-5761], Frederik Gailly [1,3][0000-0003-0481-9745], Geert Poels[1,3][0000-0001-9247-6150]

[1]Department of Business Informatics and Operations Management, Ghent University
[2] KRDB Research Centre for Knowledge and Data, Free University of Bozen-Bolzano
[3] FlandersMake@UGent – core lab CVAMO
thomas.derave@UGent.be, tiago.princesales@unibz.it, {frederik.gailly, geert.poels}@UGent.be

Abstract. In this paper a method is proposed for agile digital platform prototype development based on organization-specific ontologies. The resulting prototypes act as minimum viable product of the digital platform that is described by the ontologies. Our method combines the strengths of agile practices, to speed up the development process in a user-oriented manner, with the strengths of ontology-driven development, improving the software structure, single terminology, and communication between stakeholders. The method is demonstrated for the development of the android application 'SafaRide', a digital marketplace for safari ride sharing.

Keywords: Digital Platform, Digital Marketplace, Ontology-driven Software Development, MVP, UFO, OntoUML, DPO.

1 Introduction

The platform economy refers to activities in business, culture and social interaction that are performed on or are intermediated by digital platforms [1]. These digital platforms like Airbnb, eBay, Etsy, Ticketswap, Tinder, Dropbox and Uber intermediate in the interaction between their users. Digital platforms operating within the platform economy can be categorized by platform type [2] including multi-sided platform, digital marketplace, sharing platform, crowdfunding platform and on-demand platform. These platform types share common functions, but also have substantial differences in functionalities offered and also differ in the type of business model that is supported.

Software development and especially the development of web-based applications is a multidisciplinary and difficult task, time-consuming and highly sensitive to human interaction and team work [3, 4]. Due to the complexity in the platform economy domain, developing platform software that offers the right functionality for the in-

tended digital platform is challenging. Nevertheless, this may be minimized using an efficient software development methodology [4]. Therefore, developers adopted agile approaches that offer fast feedback, are more client-focused, capitalize on continuous improvement, and build on cross-functional teams. For agile prototype deployment, it is advised to launch a Minimum Viable Product [5], or in this case Minimum Viable Platform (MVP) [6], fast and efficient. An MVP is a product with enough features to validate the digital platform idea in an early stage of the development cycle. Existing SaaS tools for developing an MVP, like Sharetribe Go [7] which supports the development of digital marketplaces and Ever Demand [8] which supports the development of on-demand platforms, have the advantage that a developer doesn't need to start from scratch and the MVP can be developed in just a few hours. Unfortunately, these SAAS tools only focus on one specific digital platform type and do not consider the full diversity within the platform domain. Besides, they do not offer enough flexibility to develop a tailer-made MVP to the needs of the digital platform initiative. Furthermore, only a limited number of business model choices are configurable using these tools.

A solution to improve the communication between digital platform initiators and software developers and thus fasten the development of a tailer-made and satisfying MVP could be by using an 'organization-specific' ontology, which is an ontology that describes a specific existing or intended digital platform [9]. In this paper we propose a method for ontology-driven MVP development in the digital platform domain. This method was constructed using the Design Science Research Method (DSRM) of Peffers et al. [10]. Our method uses the Digital Platform Ontology (DPO) [2, 11] and continues on the research of [12] who developed a method for ontology-driven user story development, and the work of [13–15] who developed a method for ontology-driven (relational) database design. We demonstrate the proposed method with the development of an MVP running on Android called 'SafaRide' that intermediates in jeep ride sharing on a safari trip. SafaRide can be categorized as a digital marketplace, as it targets two different types of users and enables transactions between the user of both sides. This makes SafaRide a good case-study to show the advantages of ontology-driven MVP development. In this paper, we propose and demonstrate a first version of our method. In future research, we plan to apply and evaluate our method on a diverse set of digital platforms operating different business models.

This paper will proceed as follows. In section 2 we briefly present the Digital Platform Ontology (DPO), its use in developing organization-specific ontologies for digital platforms and briefly discuss the research process of how our method is constructed. In section 3 we propose our method for ontology-driven MVP development. In section 4 we demonstrate our method on the development of the SafaRide MVP. In section 5 we discuss future work, and eventually we conclude in section 6.

2 Previous Research

In the platform domain there was till recently no existing domain ontology that could be reused and no clear framework to avoid developing an organization-specific ontol-

ogy (i.e., specific to a particular digital platform) from scratch [16]. This gap was filled by (1) the development of a domain ontology, the Digital Platform Ontology (DPO) which accommodates different digital platforms types [2] , (2) a Business Model (BM) extension to the DPO (i.e., Extended DPO) which makes it easier for developers to analyze the influence of business model decisions on the creation of the platform software [11] and (3) a method for developing an organization-specific ontology [9]. Such organization-specific ontology is the result of reusing and combining classes, relationships and constraints of the extended DPO to describe a specific instance of a digital platform for platform software development purposes. Figure 1 represents the organization-specific ontology for SafaRide which is the result of reusing and combining classes, relationships and constraints of the extended DPO specific for the SafaRide business case.

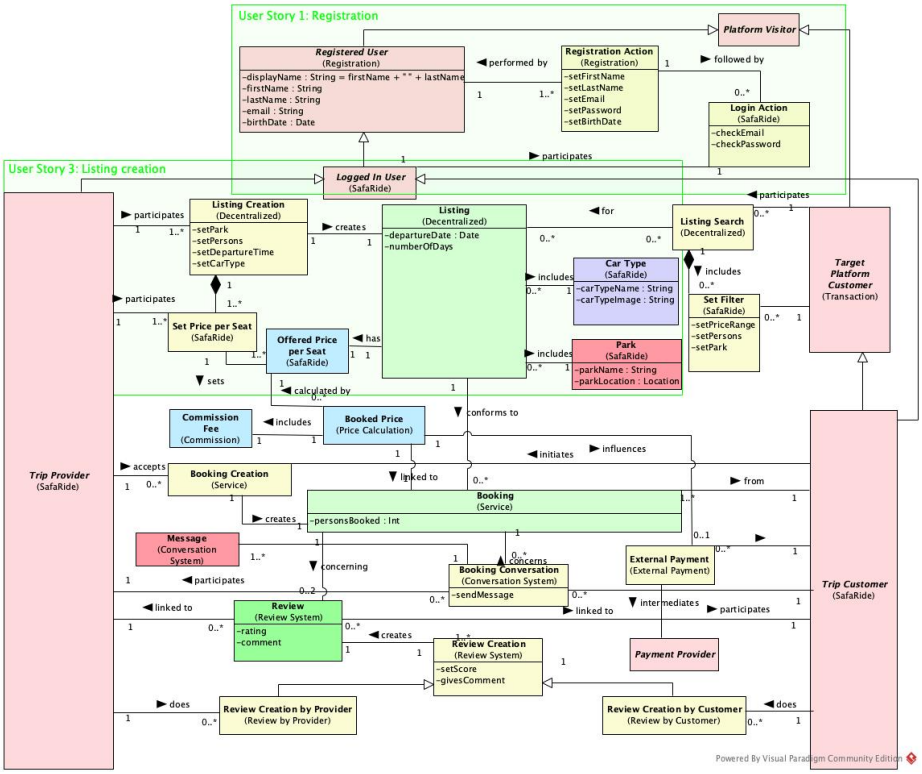

Fig. 1. Organization-specific Ontology of SafaRide

The ontology of SafaRide shows that SafaRide intermediates between trip providers and trip customers for a one-time offline service (a free seat in a safari car). After registration, a logged-in user can create a listing specifying the departure time, park, type of car and offering price per seat. Afterwards, another user (called the target platform customer) can search through the listings created using the filters and initiate a booking creation becoming a trip customer. In case the provider accepts this booking creation, the booking comes into existence capturing the booking price including a

commission fee. The booking price is transferred via an external provider, and the software allows a conversation via messages between the two users after the booking. After the delivery of the service both a review by the provider and by the customer towards each other are allowed.

In this paper an additional step in this research project is taken by designing a method for the development of an MVP starting from the organization-specific ontology for that platform. The main objectives of the proposed method are improving the shared understanding of the terminology and functionality during the development of an envisioned digital platform, decrease the perceived complexity of MVP development, improve the quality of the requirements, and improve the flexibility during development. The proposed method combines our previous research with some existing methods that use ontologies in the context of agile software development [12–15]. The paper describes the research process of how this method was designed, gives an overview of the steps within the method and finally demonstrates the method with the development of the SafaRide MVP android application.

3 A method for ontology-driven MVP development

Our method developed following the DSRM of Peffers et al. [10] integrates the methods and guidelines of digital platform organization-specific ontology development by [9], ontology-driven user story development by [12], process modeling based on user stories by [17], and ontology-driven database design by [13–15], and adds UI prototyping and MV* software design as additional elements for MVP development. An overview of our method is given in figure 2 and includes four main steps: conceptualization, analysis, MVP development and testing.

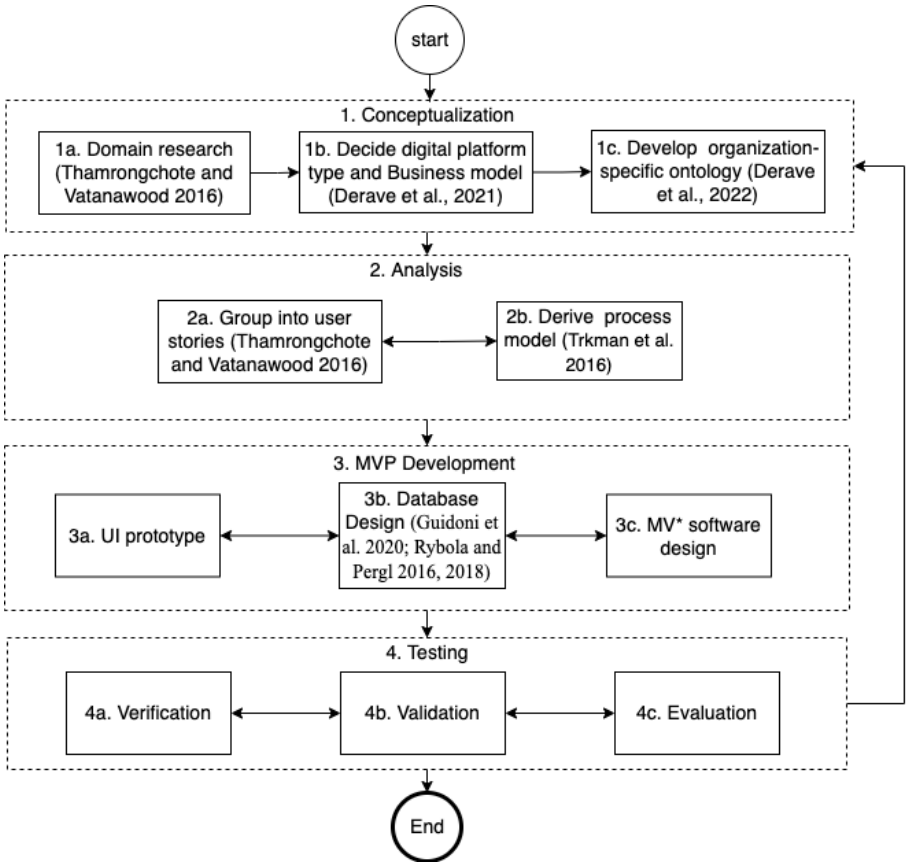

Fig. 2. Method for ontology-driven MVP development

1. First, the developers and other platform stakeholders need to conceptualize the idea of what they want to accomplish. This conceptualization is done in three sub steps.
   a. The developers need to understand the domain, the goal(s) and the added value of the envisioned software [12]. For this, significant research efforts might be required just to harmonize the requirements, concepts and terminology [3]. If the project had a previous state, historical project data can be collected [12]. But when no historical data is available, it is possible to conduct brainstorm sessions with different stakeholders.
   b. This domain knowledge is required for the choice of the digital platform type(s) using the typology of [2], and business model of the desired digital platform using the business model taxonomy of [11]. The choice of platform type and business model will influence the relevant ontology modules, and eventually shape the organization-specific ontology modules.
   c. Based on the digital platform type and business model, the developers can reuse and combine the DPO ontology modules that describe parts of the business model that the envisioned platform will deploy into an organization-specific on-

tology as explained in section 2. This ontology now captures the user roles, required functionality and other domain knowledge of the desired digital platform.

2. After, the developers need to analyze the organization-specific ontology and group the classes and relationships into user stories and a process model that further guides the MVP development process.
   a. User stories are a simple narrative illustrating of user goals that a software function will satisfy [18], and articulated in the form of 'As a [role], I want [goal], so that [benefit]'. With [role] specifying a type of user, [goal] describing the (inter)actions that the user wants the software to support, and [benefit] motivating the expected functionality from the user's standpoint. Besides writing them in text, it is also possible to use an object-oriented language like OntoUML for user stories writing [12]. The user stories of the envisioned MVP are already captured in the organization-specific ontology where a certain user role (in red) participates in an event class (in yellow) to create a certain social construct or relator class (in green) between himself and another user or the platform organization. Therefore, grouping a user role, the participating event and the created relator within a separate model grasps the user story while keeping the object-oriented presentation and ontological knowledge within the OntoUML model intact.
   b. The event classes (in yellow) in the organization-specific ontology can be reordered in a separate process model following the guidelines of [17] to visualize the happy path or functionality of a single user through the envision MVP software.
3. Within the agile philosophy, it is recommended to develop the user stories in order of importance with the development of the database, back-end (server software) and front-end (UI) software in parallel [19]. Therefore, the MVP should be incrementally developed during sprints of a selection of user stories in three non-sequential steps, with the organization-specific ontology representing the envisioned digital platform.
   a. Design a User Interface (UI) prototype (e.g., in MarvelApp or Figma). A prototype demonstrates the basic UI functionality of the platform idea before building the final version and is a fundamental part of the product design. It is possible to demonstrate the prototype to stakeholders as this helps in understanding user behavior [20]. Our experience learns that a prototype application screen or web component is required for each event class in the ontology as the prototype needs to capture the intended actions of the users. Eventually, the prototype should give a clear indication of the flow, the look and feel of the envisioned application. It is advised to do an intermediary validation of the UI prototype with potential users and other stakeholders before continuing.
   b. Ontology-driven database design is already described in a series of papers by Rybola and Pergl [13–15]. The database stores and retrieves user, listing, booking and other information in a structural way and because of the object-oriented nature of OntoUML models the organization-specific ontology easily guides the relational database design. The database development is in parallel with the UI

prototype and MVP software to guarantee a complete integration of data, information, user functionality and interface.

c. An MVP both requires a backend connecting the application to the database to store and retrieve data, and a UI frontend to interact with the user. For web applications this is typically accomplished using a Model-View-Whatever (MV*) software design pattern [21] that makes code easier to maintain and test with better user experience. The term MV* represents a family of browser-based frameworks that provide support for achieving a separation of concerns in the application's code base. The * in MV* can stand for Controller (MVC), View-Model (MVVM) or Presenter (MVP) and can be designed by many popular frameworks for application development (Android using Kotlin, Angular using Typescript, WebObjects using Java, Django using Python, Rails using Ruby, .NET using C# and other languages, Flutter using Dart, React using JavaScript, Vue.js using JavaScript). More information on how the organization-specific ontology influences each component of the MV* software design pattern is given during the demonstration of our method in section 4.

4. The last step tests the developed software and includes three non-sequential sub steps named verification, validation and evaluation [4].
   a. Verification is the demonstration of consistency, completeness, and correctness of the MVP. Therefore, we use UI tests, integration tests, unit tests and verify if the goal and benefit of each user story is fully integrated in our MVP software.
   b. Validation is the determination of the satisfaction of the MVP considering user needs and requirements. This can be accomplished by letting the users interact with the UI prototype and MVP software, to make sure the functionality and look and feel is sufficient to their needs.
   c. At last, the goal of the evaluation process is to access the quality, usability, and utility of the MVP from the point of view of those participated in knowledge acquisition phase. This is accomplished by demonstrating the organization-specific ontology, process model, UI prototype and MVP towards the management, financers and other non-user stakeholders. Their feedback will influence the next development iteration and can even adjust the digital platform type and business model of the desired MVP.

Our method has an user-oriented, iterative character as we follow an agile way of development. Through the iterative development process the organization-specific ontology constantly evolves, as flexibility of requirements is a must for agile software development projects [18].

4 Method demonstration: SafaRide

The envisioned android application for SafaRide is meant for someone who rents a safari car and still has empty seats available, but also for travelers traveling with few and looking for an already booked car to share the ride. Both types of users can be considered as 'peers' or 'prosumers' setting SafaRide within the digital marketplace domain following the definition of [22]. On top of that, the application intermediates

in the rental of an under-utilized good (free car seats), also setting SafaRide within the sharing platform domain. The added value for these peers is lower costs and the social advantage of traveling together, creating a win-win situation. The idea of ride sharing during a safari trip is brand new, and no historical data concerning safari trips was available. Therefore, we conducted brainstorm sessions with all stakeholders (in our case the four developers of the application and one African travel expert) to align the idea behind SafaRide. The conceptualization step includes the development of the organization-specific ontology of SafaRide which is already discussed in section 2.

4.1 Analysis

We use the object-oriented user story method of [12] to capture role-event-relator patterns within the relationships and classes of our organization-specific ontology[1]. As an example, we discuss two user stories grouped within the organization-specific ontology of figure 1: user story 1 – Registration and user story 3 – Listing creation. In user story 1, a platform visitor can perform a registration action to become a registered user. Only a registered user can perform a login action that enables the creation of listings and bookings. In user story 3, a trip provider can create a listing and set a price per seat, car type, safari park, departure date and number of days within that listing to facilitate a customer finding it during a future listing search.
The events within an OntoUML ontology can also be envisioned as a user activity process using a process model language (e.g., Business Process Model and Notation, BPMN). By placing the event classes (in yellow) within the organization-specific ontology in the right sequence after each other, the happy path from registration until review can be derived. Figure 3 gives part of the process model[2] capturing the event classes within user story 1 and 3. Because a logged-in user can choose the role she wants to play, an OR-gateway was needed to visualize the actions a user of each role can perform.

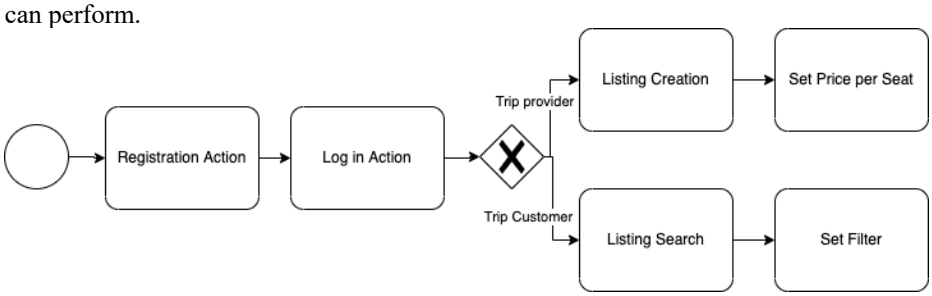

Fig. 3. Part of the SafaRide user process model for registration and listing creation

[1] A complete overview of the user stories can be found on https://model-a-platform.com/safaride-user-stories/.
[2] The complete BPMN model of SafaRide can be found on https://model-a-platform.com/safaride-bpmn-model/

4.2 MVP development

We developed the UI prototype using the prototype software 'MarvelApp'[3]. The UI prototype mainly visualizes the flow, look and feel of the envisioned SafaRide software, but doesn't capture the database design, user roles and functionality. For each user activity in figure 3, a prototype screen is designed.

In parallel, we constructed a relational database using MySQL as this is still the most popular type of data storage [13]. We copy-pasted the organization-specific ontology into a separate database model and followed the one table per hierarchy approach [15], lifting all relationships and attributes of the child classes into their parent class. For SafaRide, the registered user, trip provider, target platform customer and trip customer attributes and relationships were captured into the parent class called 'User'. After, we only keep the object classes (in red), relator classes (in green) or type classes (in purple) required for data collection and storage. For SafaRide, this was the case for user, listing, car type, park, message, booking and review. We added the mode classes (in blue) as attributes in the related object or relator classes and added indirect relationships between classes through events (e.g., user has a one-to-many relationship with listing through the listing creation event). Finally, we converted the OntoUML model into simple Unified Modeling Language (UML) notation, adding primary and foreign keys to specify the relationships while keeping the multiplicity constraints intact. If required, extra tables need to be included to solve many-to-many relationships, and tables originated from type classes with only one attribute can be included as enumeration types, but this was not the case for the SafaRide model. The final database schema in UML used for the construction of the MySQL database of SafaRide is represented in figure 4.

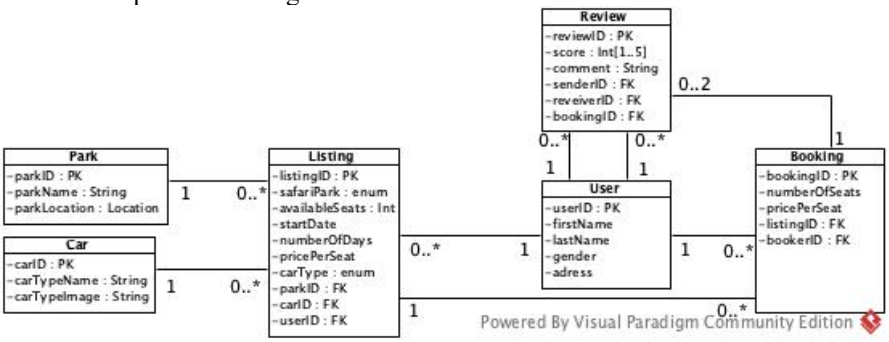

Fig. 4. SafaRide relational database schema in UML

The main contribution of this paper is the improvement of both the back-end and front-end MVP development based on the organization-specific ontology. SafaRide[4] was developed for android[5] using the Kotlin programming language with a Model-

[3]The UI prototype of SafaRide can be found on https://marvelapp.com/prototype/80ha0ha

[4] The latest version of the app can be found on http://model-a-platform.com/safaride-versions/

[5] For a guide to android app architecture: https://developer.android.com/jetpack/guide#separation-of-concerns

View-ViewModel (MVVM) design pattern. An overview of the MVVM design pattern and its components for user stories 1 and 3 of SafaRide is given in figure 5.

First, the Model is the application's dynamic data structure, independent of the UI. It is connected to the database(s) and directly manages the data, logic, and rules of the application. For the SafaRide android application, only a local data source, the MySQL database is used. For each table in our database schema (figure 4), a data class and repository are created. The main purpose of a data class is to hold data, and no functions are created within the class body as the database fields are used as parameters in the primary constructor. A repository on the other hand provides a clean API for data access to the rest of the application, independent of the database system. It reverses the records in the database to objects within the android application.

Next, the View is represented in a number of view components, and enables the user functionality of the software. Therefore, the event classes within the organization-specific ontology capture the required view components of the intended software. The View in android includes fragments that represent a reusable portion of the app's UI, and activities that are mainly used to construct a single screen of your application [23]. For the SafaRide Android application, the View includes a separate UI folder with an activity or fragment file (in Kotlin) and a layout file (in XML) for each event class in the organization-specific ontology.

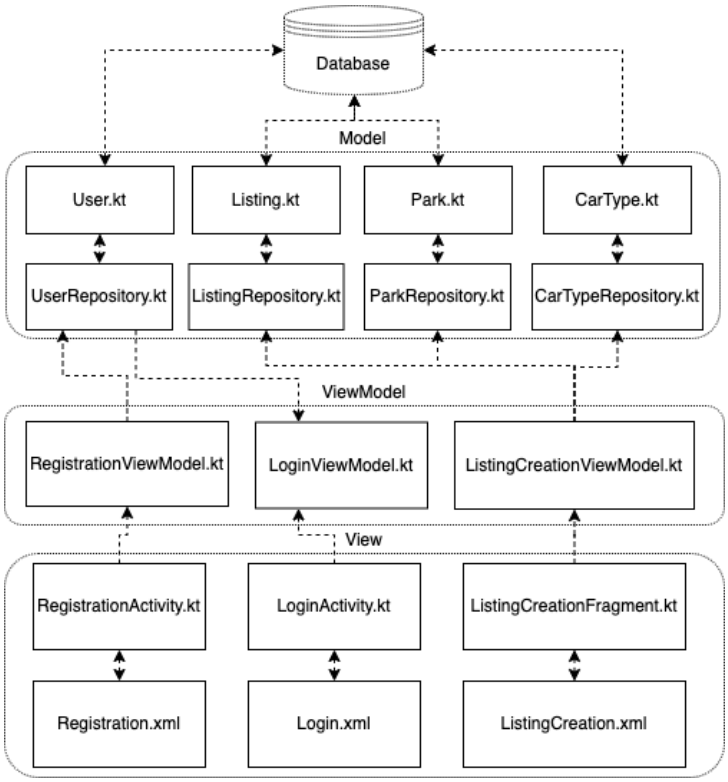

Fig. 5. SafaRide MVVM design pattern of user story 1 and 3

Finally, the ViewModel provides triggering events for changing the state of the Model and the View. This is captured in the relations between the object classes and the event classes within the organization-specific ontology. For the SafaRide Android application, a ViewModel file (in Kotlin) is created within each UI folder and connects the View to the right repositories.
Clear terminology and naming conventions during the MVP development are important; therefore, it is advised to name all classes, variables, and parameters according to the classes in the organization-specific ontology. Good variable names makes the code easier to understand and improves the development [25]. An overview of the conversions from the two user stories in figure 1 to SafaRide MVP software is given in table 1.

Table 1. Conversions from organization-specific ontology to MVP software

Class	**MVP Software**
Platform Visitor, Registered User, Logged-in User, SafaRide Trip Provider	**Model:** A User data class to define the user objects and a user repository to connect the application to the user table is created.
Registration Action	**View:** A registration XML file and registration activity is created **ViewModel:** A registration ViewModel file transfers the user data from the registration activity towards the registration repository
Login Action	**View:** Login XML file and login activity is created **ViewModel:** A login ViewModel checks the username and password with the relevant fields of the database.
Listing, Park, CarType	**Model:** A Listing, Park and CarType data class and repository are created to define the objects and connect the application to the tables within the database
Listing Creation, Set Price per Seat,	**View:** ListingCreation XML file and listingCreation fragment is created. **ViewModel:** A listingCreation ViewModel transfers the listing data from the listingCreation fragment towards the listing, park and carType repository. This includes the offered price per seat of the listing.

Of course, the organization-specific ontology doesn't include all knowledge needed to develop the MVP. Nevertheless, it structures the more complex relationships between different concepts (listing, booking, user roles), improving the efficiency of the development process. The MVP of SafaRide doesn't yet include the commission, payment, booking conversation and review functionality, as these user stories were considered as less urgent and will be developed in future development cycles following our method.

4.3 Testing

During the development of the SafaRide MVP, we designed several UI tests, integration tests and unit tests to assure the quality of the MVP software and verified the

completeness of the software with each user story. We also regularly validated the usability of the UI prototype and application with several potential users who are familiar with safari holidays, and evaluated the MVP by demonstrating the organization-specific ontology, process model, UI prototype and MVP towards the African travel expert, and taking his feedback into account during the following development iteration. The SafaRide organization-specific ontology was modified after each sprint to keep the model in line with the (intended) software structure.

5 Discussion and Future Work

In this paper, we proposed a first version of our ontology-driven MVP development method demonstrated by the development of one MVP. Besides being ontology-driven, our method follows an agile approach focusses on the development of an MVP. Software development is known as a complex activity that is highly sensitive to human interaction and team work [3]. Therefore, an agile approach requires fast feedback, is user-focused with continues improvements and cross-functional teams. Our method only considers the happy path of the user process to launch an MVP as fast as possible, with regular validations by users and other stakeholders. However, an agile approach also has a considerable number of downsides. First of all, there is also a fragmented output as teams work on different user stories without a clearly described finite end of the project [26]. Another issue is that teams can work on different user stories with a widespread use of overlapping terminology and conflicting constraints for the components, user roles and functionality of the intended software [9]. On top of that, there is a limited amount of documentation as software companies rapidly develop prototypes without saving complete information or insights acquired in a structured semantic format [27]. Ontology-driven MVP development solves these issues as the organization-specific ontology clearly captures the user stories and their interconnectedness. It describes the boundaries of each user story, and what is required from the MVP before the project ends. It aligns the terminology, improving the communication between teams working on different user stories and helps in understanding how a certain user story fits within the complete project. Besides, the ontology documents the MVP development in an easy, structural, and flexible manner. By regularly updating the requirements within our ontology throughout the development process, our method supports easy comprehension of the project's nature and makes it easier for software reuse in future projects.

In future research, we plan to validate and further improve our method by supporting the development of a diverse set of MVPs of different platform types operating a variety of business models. A test case will be set up with aspiring entrepreneurs who plan to develop an MVP of their platform idea originated from a self-constructed, DPO-based organization-specific ontology. During the development process, the version and improvements of each iteration will be monitored using GitHub classrooms, to visualize and analyze the influence of ontology modifications on the eventual MVP software. In the end, the efficiency and perceived usefulness of our method will be quantified with a questionnaire towards the software developers. Both single develop-

ers and teams are composed for the MVP development, to test the efficiency and communication improvements of our ontology-driven approach.

6 Conclusion

In this paper, we proposed a method for ontology-driven, Minimum Viable Platform (MVP) development, constructed using the Design Science Research Method (DSRM) of Peffers et al. [10]. An ontology is not only the collection of concepts, terms, constraints and relationships but also the formal, explicit, conceptual model of object ranges in a computational representation [3]. Our method is ontology-driven, as it uses an organization-specific ontology [9] based on the Digital Platform Ontology (DPO) written in OntoUML as a basis during the development process. A normal UML model only makes distinction between the data classes, while OntoUML models also capture the difference between objects events, social or financial benefit for each user. In the organization-specific ontology, objects and relators portray the required data structure, while events portray the required functionality of the intended software [9]. Therefore, the organization-specific ontology can be divided in user stories with each user story describing a user role, what the user of this role wants and how he benefits from that. The ontology captures the required functionality of these user stories and transformations between the organization-specific ontology and the code are used to construct the final software.

A clear method for MVP development is important, because due to high costs and duration of the project [28], competitors with less diversification but a superior technology are still capable to monopolize a market [29]. Lowering the barrier of digital platform development is therefore vital, as many existing platforms have the tendency to apply a 'winner-takes-all" strategy to create a monopoly. An essential element that creates incentives to enter and isolate the influence of competitors is increasing the differentiation of digital platforms. This way, network effects are mitigated, and divide-and-conquer strategies are less effective, which reduces the monopolization problem at the same time [29].The proposed method helps to increase the knowledge of digital platform design, which triggers the conception of alternatives for monopolistic companies such as Airbnb and Uber, who are criticized for paying low wages, taking high commission fees, and avoiding taxes [30]. This may facilitate the development of diverse, smaller, more alternative, and socially responsible platforms and thus contribute to the creation of a more socially responsible platform economy.

References

1. Kenney, M., Zysman, J.: The rise of the platform economy. Issues Sci. Technol. 32, 61–69 (2016).
2. Derave, T., Sales, P.T., Gailly, F., Poels, G.: Comparing Digital Platform Types in the Platform Economy. In: Caise 2021. pp. 5–10 (2021).
3. Clarke, P., Mesquida, A.L., Ekert, D., Ekstrom, J.J., Gornostaja, T., Jovanovic, M., Johansen, J., Mas, A., Messnarz, R., Villar, B.N., O'Connor, A., O'Connor, R. V.,

Reiner, M., Sauberer, G., Schmitz, K.D., Yilmaz, M.: An investigation of software development process terminology. Commun. Comput. Inf. Sci. 609, 351–361 (2016).
4. Hasan, S.S., Isaac, R.K.: An integrated approach of MAS-CommonKADS, Model-View-Controller and web application optimization strategies for web-based expert system development. Expert Syst. Appl. 38, 417–428 (2011).
5. Ries, Er.: The Lean Startup. Currency (2011).
6. Gracia, C.: Your marketplace MVP – How to build a Minimum Viable Platform, https://www.sharetribe.com/academy/how-to-build-a-minimum-viable-platform/#:~:text=A Minimum Viable Product (MVP)—or%2C in the,both sides of the marketplace.
7. Sharetribe: Sharetribe Go, https://github.com/sharetribe/sharetribe, (2019).
8. Ever Corporation: Ever Demand, https://github.com/ever-co/ever-demand, (2022).
9. Derave, T., Sales, T.P., Gailly, F., Poels, G.: Sharing Platform Ontology Development : Proof-of-Concept. Sustain. 1–19 (2022).
10. Peffers, K., Tuunanen, T., Rotherberger, M.A., Chatterjee, S.: A Design Science Research Methodology for Information Systems Research. J. Manag. Inf. Syst. 24, 45–78 (2008).
11. Derave, T., Sales, T.P., Gailly, F., Poels, G.: Understanding Digital Marketplace Business Models : An Ontology Approach. In: POEM. pp. 1–12 (2021).
12. Thamrongchote, C., Vatanawood, W.: Business process ontology for defining user story. 2016 IEEE/ACIS 15th Int. Conf. Comput. Inf. Sci. ICIS 2016 - Proc. 3–6 (2016).
13. Rybola, Z., Pergl, R.: Towards OntoUML for software engineering: Optimizing kinds and subkinds transformed into relational databases. Lect. Notes Bus. Inf. Process. 332, 31–45 (2018).
14. Rybola, Z., Pergl, R.: Towards OntoUML for software engineering: Transformation of Anti-rigid sortal types into relational databases. Proc. 2016 Fed. Conf. Comput. Sci. Inf. Syst. FedCSIS 2016. 1581–1591 (2016).
15. Guidoni, G.L., Almeida, J.P.A., Guizzardi, G.: Transformation of Ontology-Based Conceptual Models into Relational Schemas. Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics). 12400 LNCS, 315–330 (2020).
16. Mohamad, U.H., Ahmad, M.N., Zakaria, A.M.U.: Ontologies application in the sharing economy domain: a systematic review. Online Inf. Rev. ahead-of-p, (2021).
17. Trkman, M., Mendling, J., Krisper, M.: Using business process models to better understand the dependencies among user stories. Inf. Softw. Technol. 71, 58–76 (2016).
18. Sh Murtazina, M., Avdeenko, T. V.: The ontology-driven approach to support the requirements engineering process in scrum framework. CEUR Workshop Proc. 2212, 287–295 (2018).
19. W3schools: What is Full Stack?, https://www.w3schools.com/whatis/whatis_fullstack.asp.
20. marvel: A guide to creating your first prototype, https://help.marvelapp.com/hc/en-us/articles/360002536038-A-guide-to-creating-your-first-prototype#:~:text=A prototype demonstrates the functionality,also do in Marvel!).

21. Emmit, A.S.J.: SPA Design and Architecture: Understanding single-page web applications. Manning (2015).
22. Täuscher, K., Laudien, S.M.: Understanding platform business models : A mixed methods study of marketplaces. Eur. Manag. J. 36, (2018).
23. geeksforgeeks: Difference Between a Fragment and an Activity in Android, https://www.geeksforgeeks.org/difference-between-a-fragment-and-an-activity-in-android/.
24. Srivastava, V.: MVC vs MVP vs MVVM architecture in Android.
25. Minnick, C., Holland, E.: Naming JavaScript Variables, https://www.dummies.com/web-design-development/javascript/naming-javascript-variables/.
26. Lynn, R.: Disadvantages of Agile, https://www.planview.com/resources/articles/disadvantages-agile/, (2020).
27. Adnan, M., Afzal, M.: Ontology based multiagent effort estimation system for scrum agile method. IEEE Access. 5, 25993–26005 (2017).
28. Handgraaf, S.: Five ways to build an online marketplace platform—and how to choose yours, https://www.sharetribe.com/academy/ways-build-marketplace-platform/.
29. Sanchez-Cartas, J.M., Leon, G.: Multi-sided Platforms and Markets: A Literature Review. SSRN Electron. J. 1–62 (2019).
30. Kenney, M., Zysman, J.: Choosing a future in platform economy: The Implications and Consequences of Digital Platforms. J. Chem. Inf. Model. 53, 1689–1699 (2013).