B-MERODE: A Model-Driven Engineering and Artifact-Centric Approach to Generate Blockchain-Based Information Systems

Victor Amaral de Sousa[1(✉)], Corentin Burnay[1], and Monique Snoeck[2]

[1] University of Namur, Namur, Belgium
{victor.amaral,corentin.burnay}@unamur.be
[2] KU Leuven, Louvain, Belgium
monique.snoeck@kuleuven.be

Abstract. Blockchain technology has the potential to facilitate the development and improvement of cross-organizational business processes. When it comes to developing systems relying on blockchain having this purpose, model-driven engineering is a promising approach that has been adopted by a number of solutions. This paper presents B-MERODE, a novel approach relying on model-driven engineering and artifact-centric business processes to generate blockchain-based information systems supporting cross-organizational collaborations. The feasibility of the approach is demonstrated by modeling the case of a rice supply chain. Compared to other solutions, B-MERODE provides more flexibility, more reusability and further leverages the automation potential offered by model-driven engineering.

Keywords: Blockchain · BPM · MDE · MERODE · Method

1 Introduction

With the availability of blockchain technology and its capability to run software (often referred to as smart contracts), a broad range of applications have been made possible. Among these are opportunities for the implementation, execution and monitoring of cross-organizational Business Processes (BPs), as further detailed in [1]. A blockchain can be described as "[…] a distributed database, which is shared among and agreed upon a peer-to-peer network. It consists of a linked sequence of blocks, holding timestamped transactions […] secured by public-key cryptography […]. Once an element is appended to the blockchain, it cannot be altered, turning a blockchain into an immutable record of past activity" [2, p. 3].

However, being a new technology, developing solutions relying on it remains a challenging task, for multiple reasons mentioned in [3]. These include the lack of adequate developer tools, the lack of people having the required skills and knowledge and a steep learning curve.

One of the promising approaches that can facilitate the development of Blockchain-Based Information Systems (BBISs) supporting cross-organizational collaborations is

S. Dustdar et al. (Eds.): CAiSE 2020, LNCS 12127, pp. 117–133, 2020.
https://doi.org/10.1007/978-3-030-49435-3_8

Model-Driven Engineering (MDE). MDE enables generating executable code from a set of models that specify the processes to be supported. For example, [4] and [5] are tool-supported methods that allow to generate Solidity BBISs respectively from BPMN diagrams and Finite State Machines (FSMs). Using MDE requires a language to create models used as input for code generation. Without insisting on MDE, authors of [6] call for the development of a shared ledger business collaboration language. This should be a language that is understood by business people and which can be used to specify cross-organizational BPs supported by blockchain technology. Such a language would foster blockchain adoption and facilitate prototyping. Furthermore, [6] argues that the language should be based on artifact-centric BP models. In the past, several approaches to model-driven blockchain-based cross-organizational BPs have been proposed. Many of those are however process-centered rather than artifact-centered. Even though a few artifact-centered approaches exist, these provide only a partial coverage of the required concepts, the code generation is not fully automated, and they impose a number of constraints to developers, hereby reducing reusability and flexibility.

As a response to the need to develop new methods for the analysis and engineering of BPs supported by blockchain (detailed in [1]), the present paper proposes B-MERODE. B-MERODE consists of a method and an artifact-centric shared ledger business collaboration language that can be used as input to generate BBISs using MDE. In the remainder of the paper, we use the term BBIS to refer to software running on the blockchain (often referred to as smart contracts). The proposed method aims to alleviate or even resolve the shortcomings of existing artifact-centered approaches. Our proposition is based on MERODE, a method for the design and implementation of intra-organizational Enterprise Information Systems (EISs). After having designed B-MERODE, we use it to model a rice supply chain case as a first validation of the feasibility of the approach.

The remainder of the paper is organized as follows. Section 2 introduces the research problem by first discussing requirements for solutions aiming at generating BBISs to support cross-organizational collaborations. Then, a review of the related work is proposed and analyzed in the light the identified requirements. Based on this, a number of improvements are suggested and the basic ingredients for a new solution allowing to realize such improvements are discussed. After that, Sect. 3 presents our solution called B-MERODE. To have a first validation of the feasibility of the proposed approach, Sect. 4 presents the case of a rice supply chain modeled using B-MERODE. Finally, Sect. 5 discusses the limitations of our approach and suggests further research leads.

2 Research Problem

2.1 Requirements

When developing blockchain-based solutions to support cross-organizational BPs, a number of requirements (both functional and non-functional) have to be met:

- Dynamic Participants Management [7]: to handle the dynamic nature of cross-organizational BPs, it is key to have the ability to change participants as the process is running (e.g. replacing a wholesaler by another in case the first becomes unavailable).

- Minimal Information Sharing [7]: considering the trust issues among collaborating organizations, which may even be competitors, minimizing the information that needs to be shared among the participants is key. This information can be about the identity of the participants, the data used in the process (e.g. details about an order) and information about the internal structure and processes of the participants.
- Formal Verification [5, 6, 8]: considering the immutable character of BBISs, it is crucial to ensure the correctness and soundness of these before deploying them. Indeed, once deployed, the code cannot be changed, and errors may be unfixable and thereby lead to unresolvable exploits.
- Reusability [8]: to avoid starting every process model from scratch, it is valuable for organizations to have the ability to reuse (parts of) models previously developed.
- Flexibility: when modeling and executing BPs, flexibility is needed for many different aspects, including management of the participants at runtime (c.f. dynamic participants management); support of variants of BPs [8]; and support for unpredictable processes (i.e. processes where the sequence of activities may vary from one case to another) [9]. We further emphasize this requirement by including, as part of the flexibility needs, the possibility for involved organizations to have as much freedom as possible in the specification and implementation of internal processes. Besides elements that are agreed upon as part of collaborations, the participants should remain free to implement their internal business processes as they want and using the technology of their choice.

2.2 Related Work

In the literature, a number of solutions using MDE to generate BBISs supporting cross-organizational BPs can be found. The majority of these solutions rely on process-centric BP models to specify the processes. It has been argued that artifact-centric BP models offer a better alternative to build a shared ledger business collaboration language [6]. The process-centric paradigm focuses on activities and their sequence and only skims on data-aspects. In contrast, the artifact-centric approach focuses on Business Objects (BOs) also called business artifacts, representing key tangible or conceptual entities relevant to the business. To define the different components of artifact-centric BPs and to compare different models within that paradigm, the BALSA framework can be used [10]. It describes the concerned processes as made of *Business Objects (*or *Business Artifacts*), *Lifecycles*, *Services* and *Associations*. Every BO can be related to others and is described by a number of data attributes. Furthermore, every object has a lifecycle that defines the key, business-relevant stages that it can go through. The services are used to make changes to BOs. These changes can be a modification of the value of the attributes, or the switch to a new state in the lifecycle. Finally, associations define the conditions under which services can be executed.

As detailed in [6], artifact-centric BPs have a number of advantages when it comes to creating a shared ledger business collaboration language. Indeed, the layered approach they adopt provides improved flexibility, reusability and adaptation as well as a clearer separation of concerns. On top of that, the type of models used are argued to be more intuitive, which facilitates their creation and reasoning about them. Furthermore, these models are declarative, which further improves flexibility. Finally, the proposed models support formal reasoning, which is crucial in a blockchain context. One of the

important aspects of BPs which is not explicitly detailed in the BALSA framework is around permissions management. Another framework which incorporates the BALSA dimensions (in a slightly different way) as well as permissions management aspects is the Philharmonic flows framework [11]. However, artifact-centric BP models as described in the BALSA framework can incorporate the fine-grained access permission constructs defined in Artifact-Centric Service Interoperation (ACSI) hubs [6]. These constructs are argued to be well-suited for a blockchain-enabled cross-organizational context [6] and they are more flexible than the ones described in the Philharmonic flows framework.

Starting from the idea of using artifact-centric BP models and MDE, the present section focuses on solutions that, at least to some extent, belong to the artifact-centric paradigm. The solutions that were identified are *Lorikeet* [12], *FSolidM* [5] and a solution relying on *Dynamic Condition Response* (DCR) *graphs* [13]. The remainder of this section provides an overview of the coverage of the requirements identified in Sect. 2.1 in these solutions.

The dynamic management of participants is only explicitly supported by *Lorikeet*. In terms of information sharing, the solution which requires the least sharing is *Lorikeet*. *FSolidM* does not provide explicit support for data-related aspects and neither does the solution using *DCR graphs*. Furthermore, this solution requires sharing the identity of the participants. In terms of opportunities for formal verification, all the reviewed solutions can support it, to some extent. When it comes to reusability, it appears to be limited within every solution. Indeed, considering that for most solutions, only or mainly the activity sequence is effectively supported, this is the only perspective of processes that can potentially be reused. Finally, *FSolidM* and *DCR graphs* provide a lot of flexibility regarding many aspects of business processes. The reason for that is that they only or mainly support the activity sequence. The fact that aspects related to data, permissions and participants are mostly left aside by existing solutions strongly limits their reusability but also their supported flexibility. By supported flexibility we mean flexibility for aspects which are explicitly supported.

While the reviewed solutions can be associated with the artifact-centric paradigm, it appears that they do not adopt it to an extent allowing to reap the most benefits out of it. Mainly, the separation of concerns into different layers and the management of data and permissions could be further improved with a method that follows more closely the considered paradigm.

Considering this, we call for a solution which offers explicit support of BP aspects related to data, permissions, participants and activity sequences. Furthermore, we call for a solution that provides flexibility and reusability in these aspects. By supporting more aspects of BPs, the power of MDE can be further leveraged, and its return on investment further increased.

2.3 Ingredients for an Improved Solution

To create a solution which meets the objectives mentioned in the previous section, we rely on three foundational ingredients: MDE, artifact-centric BPs (c.f. Sect. 2.2) and MERODE. The present section introduces MERODE, motivates its choice and discusses its limitations in a context where the goal is to generate BBISs supporting cross-organizational BPs.

Introduction to MERODE. MERODE is a method that relies on MDE and artifact-centric BPs to design and implement intra-organizational EISs. The Platform-Independent Model (PIM) consists of a number of models that are precise and complete enough to allow for the automatic generation of an executable application from them. The method is supported by tools for creating and verifying models and for the generation of working prototypes from these models. The method also provides methodological guidelines for ensuring the quality of the EIS's design.

MERODE considers EISs as being made of three distinct layers: the domain layer, the Information System Services (ISSs) layer, and the Business Process (BP) layer. The first one describes Business Object Types (BOTs), their respective data attributes and the relationships among different object types. Examples of BOTs could be *Customer* and *Order*, with each order being related to a customer. As part of the domain layer, the lifecycles of the different BOTs are also described.

The ISSs layer describes the services offered by the software, divided in two types: input and output services. The former can affect business objects by updating their attributes' value or the current state of their lifecycle. The output services provide (reading) access to the BOs.

On top of that layer lies the BP layer. It defines the work organization and uses the defined ISSs to interact with the domain layer, which holds information about BOs.

While MERODE uses and/or suggests particular models to represent each element of the three layers, the main part whose transformation towards a working prototype is automated (by the tools developed to this date) is the domain layer[1]. The domain layer is described by means of three inter-related models. First, the Existence Dependency Graph (EDG) specifies the BOTs, the relationships among them and their respective attributes, using a subset of the Unified Modeling Language (UML) class diagram notation. Then, the Object-Event Table (OET) is used to specify the business events that can be triggered on the BOs. When an event fires, it triggers the execution of methods on the BOs, as specified in the OET. These methods are used to create, modify or end BOs. Finally, using FSMs, the lifecycles of the BOTs are specified. The FSMs define the states in which the BOTs can be and the transitions from one state to another, which happen when methods are executed. For more detailed explanations about MERODE, the reader is referred to [14] and [15]. The first source provides an overview and summary of the method while the second describes it in much more details.

Motivations for the Use of MERODE. In Sect. 2.2, motivations to adopt the artifact-centric paradigm to model blockchain-enabled cross-organizational collaborations have been discussed. One of the motivations for adopting MERODE as a founding approach is that it belongs to the artifact-centric paradigm and provides a good coverage of the BALSA dimensions. As suggested in artifact-centric BPs, MERODE adopts a layered approach with the domain layer, the ISSs layer and the BP layer. The EDG is used to specify business artifacts, their relationships and their attributes. The OET and the FSMs are then used to define the lifecycles of the different BOTs. Besides that, the different services defined in the ISSs layer allow to cover the services dimension. Finally, the

[1] A recent extension allows defining a presentation model, and generating tailored user interfaces accordingly [23]. This extension is however less relevant in a blockchain context.

BP layer allows defining the associations. This layered approach adopted by MERODE allows for more reusability and flexibility, both part of the requirements identified in Sect. 2.1. For example, while the OET and FSMs could be different from one solution to another, it would be possible to reuse a given EDG. Going further, a complete domain model could be reused for different purposes.

Overall, MERODE uses a set of layers and a set of models, and integrates all the different models in a consistent way, providing a rather holistic specification (and possible implementation) of the system being developed considering the covered BALSA dimensions.

When using multiple models that need to be integrated, consistency across the models is key to avoid undesired behaviors. The approach proposed by MERODE offers consistency by construction [14]. It does so first at a fundamental level, when defining the different models to use and their integration. On top of that, the approach includes a number of rules to be respected, and for which compliance can be checked automatically and formally. Such consistency checking across the different models is further detailed in [15, Sect. 6.3]. Still related to consistency, some of the models used in MERODE offer opportunities for formal reasoning (e.g. formal deadlocks detection [16]), which is one of the arguments in favor of artifact-centric approaches as well as one of the requirements identified in Sect. 2.1.

Overall, MERODE is a good candidate to meet the formal verification, reusability and flexibility requirements. However, the requirement of dynamic participants management is not explicitly supported. As for the minimal information sharing needs, these would depend upon the actual models built with MERODE.

As an alternative to MERODE, other approaches also offer a good coverage of the BALSA dimensions. For instance, [17] proposes to cover all the dimensions using a defined set of UML models (not especially in a blockchain context). It suggests using class diagrams for the business artifacts dimension, FSMs for lifecycles, Object Constraint Language (OCL) operation contracts for services and activity diagrams for the associations. MERODE is compatible with the proposition made in [17] but offers a better support for code generation and more flexibility in the way services and associations are specified.

In short, MERODE provides a robust basis to model information systems in a consistent, flexible, reusable and holistic way. However, this approach was not initially focused on the design of systems supporting cross-organizational BPs. Some of the limitations of MERODE for that context are discussed in the section that follows.

Limitations of MERODE. The first identified limitation is that there is no distinction between what needs to run on the blockchain and what can be run off the chain. However, such a distinction needs to be made in a blockchain-based setting, as discussed in [19], as it has an important impact in the way particular aspects (e.g. permissions) are managed.

A second limitation is about permissions management. Whether we are in an intra- or inter-organizational setting, it is important to define who (i.e. which role or which participant) is allowed to perform which action. However, MERODE does not specify how permissions should be handled, it has no explicit notion of neither participants not

participant types and it does not explicitly support the dynamic participants management (c.f. Sect. 2.1).

These two groups of limitations call for a number of changes and extensions of MERODE to make it suitable for blockchain-enabled cross-organizational settings, which are discussed in the next section.

3 Contribution: B-MERODE

The present section introduces the adaptations that we propose for MERODE to make it suitable for blockchain-enabled cross-organizational settings, considering the limitations presented in the previous section. We thereby propose a new version of MERODE that we call B-MERODE. Figure 1 shows an overview of the different layers that we propose as well as their interactions. The elements that are depicted using dotted lines are the ones that have been added in B-MERODE or that were modified from MERODE. As further detailed in the remainder of this section, B-MERODE provides two new layers compared to MERODE: the permissions layer and the core information system services layer. In our approach, we also bring a number of modifications to the existence dependency graph and to the object-event table. The information system services and BP layers remain essentially the same as in MERODE. However, due to the distinction between what is managed on and off the blockchain, B-MERODE provides additional flexibility.

Existence Dependency Graph. In order to support the permissions layer (described in the present section) as well as other aspects of the overall solution, a number of adaptations need to be performed at the level of the EDG. These changes are the distinction between regular and participant object types as well as the distinction between base and derived attributes.

Regular and Participant Business Object Types. In the permissions layer, knowledge about the different participant types and business event types is required. Business event types have not been adapted in our approach and remain the same as in MERODE. However, to identify the different participant types, we propose an extension of the existence dependency graph. Instead of having only BOTs, a distinction is made between regular BOTs and the ones that represent participant types (as it is the case in the Philharmonic flows framework [11]). A participant type is a BOT that represents a type of entity being involved in a collaboration. For instance, in the example of [18], there were, among others, paddy batches, paddy ownerships, rice processing companies and wholesalers. The two first are regular BOTs and the two last represent different types of participants, of which multiple instances can be created (e.g. there might be multiple rice processing companies involved in a collaboration). In the end, participant object types are regular BOTs marked with a "participant" flag.

Describing the different participant types within a business network by means of the EDG provides a number of advantages. First, it allows describing the relationships among the participants, with some of them being explicitly able to influence the behavior of others. A second advantage is that it allows controlling the lifecycle of the different entities as they are involved in the collaboration. For instance, a particular process could be defined using B-MERODE models to approve a new entity to join the collaboration.

It would also be possible to dynamically add, modify and remove participants. Thereby, unlike MERODE, it allows to cover the dynamic participants management requirement identified in Sect. 2.1.

Base, Derived OCL and Derived Complex Attributes. A second modification that we propose to add is related to the attributes of the BOTs. In MERODE, every attribute is characterized by a name and by a value type (e.g. integer or string). A general distinction that can be made is between base attributes and derived attributes. Derived attributes have their value computed based on the value of other attributes. For instance, in the case described in [18], the overall quality score of a paddy batch is derived from the quality scores computed for the different steps in the process followed by the rice processing company. While MERODE never mentions what to do with derived attributes and therefore does not make an explicit distinction between how base versus derived attributes should be handled, in a blockchain context this distinction needs to be made explicitly. While it should be possible to modify the value of base attributes directly, it should not be possible for one of the participants to change the value of derived attributes directly. Indeed, the rules for the computation of a derived attribute should be agreed upon by the different participants, and it should not be possible to deviate from these rules (unless all concerned participants agree).

It is indeed not conceivable to let every participant decide how derived attributes are computed. This is due to the trust issues faced in the cross-organizational settings for which B-MERODE is proposed and which are not considered in MERODE.

Among the derived attributes, a distinction is also required. For some derived attributes, defining how the value should be computed is doable using OCL. In other cases, it is not possible or too complex. OCL allows defining the value of a given attribute based on other attributes of the same object/attributes of other objects to which a given object has access. However, there may be cases where external (e.g. web) services need to be used to retrieve some data, or where it is not possible or too complex to formulate the calculation rules using OCL. For instance, a web service could be consumed to retrieve the price of a stock at a given point in time. In a blockchain context, this means using an oracle to retrieve the data at hand. In this context, more flexibility should be provided. To do so, the designers creating B-MERODE models are allowed to write computation rules directly in the target programming language (i.e. the language in which BBIS code is generated).

So, three types of attributes are proposed in B-MERODE: base attributes whose value is manually set; derived OCL attributes whose value can be computed using OCL rules; and derived complex attributes that cannot be (easily) computed using OCL rules.

Object-Event Table. As explained in Sect. 2.3, the OET defines methods which are assigned a type of effect: creating, modifying or ending BOs. Creation and ending methods can remain defined as they are in MERODE. In MERODE no further details are given on how to specify the effects of a method. However, in a blockchain context it is required to specify at least the attributes that are (potentially) modified by the method, for two reasons.

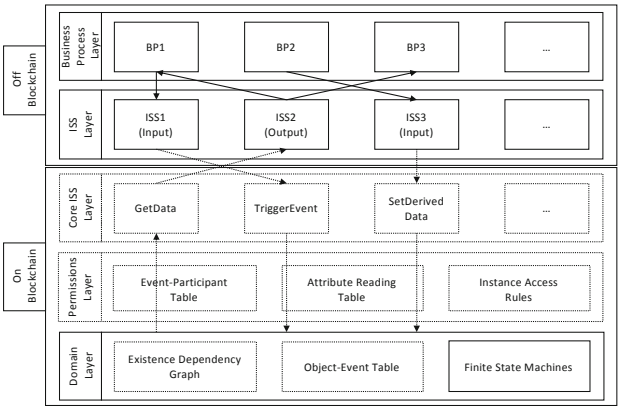

Fig. 1. Overview of B-MERODE layers

First, it is required to make a distinction between methods that will modify base attributes and the ones that modify derived attributes. Methods modifying derived complex attributes should be implemented as part of core information system services (more details on this later in the present section). Considering that to define which core information system services are required, it is necessary to know which derived complex attributes can indeed be modified, providing a list of modified attributes for a method allows performing this identification.

Second, in order to generate executable code for the modifying methods, it is required to know which parameters should be provided, and which attributes should get which of the parameters' value as a new value. Defining the attributes modified by a given method provides the ability to do so. B-MERODE thus further refines MERODE with defining the modified attributes for each method.

Finite State Machines. FSMs are defined and used in B-MERODE just as in MERODE.

Permissions. When modeling a collaboration, it is crucial to define the different participant types as well as their respective permissions. Apart from mentioning that authorizations are cross-cutting the three layers, the MERODE approach provides no further details as how to handle permissions. However, permissions are an essential element on which collaborating organizations need to agree, and should therefore not be left at the liberty of the respective participants. For this reason, we suggest the addition of a permissions layer (on top of the domain layer) that is executed on the blockchain and ensures that only operations allowed for a given participant are successfully executed. Furthermore, this allows controlling information sharing among the participants, which was identified as an important requirement. To model that layer, we rely on three permissions constructs of ACSI hubs that are compatible with artifact-centric BPs, as defined in [6]:

- *Views* can be used (among others) to define which BOTs and which attributes of these can be accessed by which type of participant.

- *CRUD operations* define which type of operation (create, read, update or delete) can be executed by which participant type on which BOT and/or attribute. In MERODE, the creation, modification and ending of BOs happen through the execution of methods that are triggered upon occurrence business events. Therefore, defining which participant type can create, modify or end a BO is determined by defining which business event type can be triggered by which participant type. The same goes for the modification of the attributes' values. For the reading of attributes' values, there is a need to specify which participant type can read which attribute.
- *Windows* define the conditions under which particular instances of BOTs (i.e. business objects) can be accessed (read or modified) by the different participants. For instance, in the case described in [18], a given *rice processing company* may be able to interact only with *RPCOwnership* objects that are assigned to that particular company. To define windows, we rely on the Access Control Language (ACL) used in Hyperledger Composer[2].

To represent the different permission constructs mentioned above, we propose a permissions model consisting of three components: the *Event-Participant Table* (EPT) defining which participant type can trigger which business event type (e.g. in Table 1); the *Attribute Reading Table* (ART) defining which attribute can be read by which participant type (e.g. in Table 2); and a set of *Instance Access Rules* (IARs).

For the IARs (e.g. in Sect. 4), five elements need to be defined: the type of operation (CRUD); the BOT that is concerned; the participant type for which a condition is specified; the condition itself (in OCL format); and finally, whether or not the operation should be allowed provided that the condition is met. Allowing model designers to choose whether the operation should be allowed provided that the OCL constraint is met provides the ability to specify rules either in a positive or negative formulation. For example, stating that a given customer can only see its own orders could be specified either as "a customer can see the orders that it has placed" or as "a customer cannot see the orders placed by other customers".

In the proposed permissions model, views, CRUD operations and windows are used. However, there is no one-to-one mapping between these permissions constructs and the models that we propose. The EPT contributes to the definition of a part of the CRUD operations, focusing on the create, update and delete operations. Reading permissions are mainly defined in the ART, which encompasses views and reading operations (part of CRUD operations). The IARs are used to represent the windows.

Core Information System Services Layer. Between the permissions layer and the ISS layer, we add another layer for *Core Information System Services* (CISSs). This layer is dedicated to the services that need to be executed on the blockchain. These include two default services: *GetData* to retrieve data from the domain layer and *TriggerEvent* to trigger events also defined in the domain layer. In addition to these services, users have to define one CISS per derived complex attribute. It will allow to modify the value of the attribute at hand, according to rules agreed upon by the different participants.

[2] https://hyperledger.github.io/composer/v0.19/reference/acl_language (accessed on 29/11/19).

Every call to a core ISS will check, through the permissions layer, whether or not the participant trying to create, read, modify or end a BO is allowed to do so. When one of the base attributes involved in a derived OCL attribute is updated, the derived attribute itself will have its value automatically updated. Therefore, additional core information system services are only required for derived complex attributes.

Information System Services Layer. In MERODE, information system services are used to interact with the domain layer. However, in B-MERODE, ISSs do not do it directly. Instead, they have to go through the CISSs layer, which will ensure that permissions are handled correctly. While the CISSs are executed and managed on the blockchain, base ISSs are not.

Using B-MERODE, collaborating organizations are free to model and implement ISSs using any technology that can interact with BBISs.

With this approach, organizations remain free to facilitate input and output in and from the domain layer as they wish and without needing to share any details about the developed services with other participants. For example, if an organization wants to combine data from the domain layer with internal data at its disposal, it can do so in an output ISS, without informing any other participant. A good example would be a service assigning tasks to individual people working for one of the participants. Indeed, the scope which is being considered in our context is the collaboration among enterprises. Therefore, while nothing prevents organizations from including details about their employees in the EDG (e.g. through an "employee" BOT), they are unlikely to be willing to share this information as part of the collaboration. Therefore, if the collaboration is kept as a scope, there is a need to manage permissions inside the organizations as well (because not every employee can do whatever he or she wants). To implement that, an ISS could be developed.

This flexibility does not come at the expense of the consistency and security of the data stored and managed on the blockchain. Indeed, every ISS needs to go through a core ISS to interact with the domain layer. Core ISSs will only execute changes/retrieve data if the operation is allowed by the permissions layer.

Business Process Layer. Finally, the top layer is the BP layer, as described in MERODE. Since it is also managed outside of the blockchain, participants are free to define their internal BPs using the tools, models and technologies of their choice without having to share any details with other participants. The different tasks in the process would use the services defined in the ISSs or core ISSs layer to interact with the domain layer.

3.1 Comparison to Related Work

Compared to other solutions, B-MERODE offers explicit support for aspects of business processes related to data, permissions, participants and activity sequence. It does so by relying on a layered approach, which allows for more flexibility and reusability. Furthermore, by explicitly covering all the aspects mentioned above, it allows to model BPs in a more holistic way. This in turn helps further leveraging the potential of MDE

and thereby further increase its return on investment. Overall, B-MERODE offers a good coverage of all the requirements identified in Sect. 2.1.

Indeed, B-MERODE allows to dynamically manage participants and goes even further by allowing users to define, directly in B-MERODE, how the network of participants should be managed. In terms of information sharing, through the permissions constructs that the method incorporates, users have support and flexibility to define the information sharing needs as they wish. Regarding formal verification, the models that are used in our method to specify aspects related to data and to activity sequences can be formally verified, as discussed in [15]. On the level of reusability, by adopting a layered approach and considering the models that are used, reusability of parts of a B-MERODE model is possible. Finally, our method offers a lot of flexibility. First, the network of participants can be dynamically managed during process execution, and the way participants can evolve during the process can be specified using the provided models. Then, variants of BPs can be supported by creating new models that reuse parts of other previously developed models. On top of that, the model chosen to represent activity sequences supports unpredictable processes and variability in the process execution. Finally, with B-MERODE, organizations remain free to develop a number of services as well as their internal business processes as they wish, using the technology of their choice, and without needing to share any information about this with other participants.

4 Validation

In order to provide a first validation of the method that we propose, this section presents an adaptation of the rice supply chain model developed in [18] using B-MERODE. In that case, the supply chain is suffering from poor performance with issues including delayed payments, poor information access as well as fraud and tampering of information, ultimately leading to higher costs for consumers. As argued in [20], blockchain can help alleviate some of these issues. The case described in [20] is first modeled using MERODE and then with B-MERODE in [18]. The present section discusses adaptations that were made to go from the MERODE model to the B-MERODE model.

Existence Dependency Graph. In the EDG proposed in [18, Fig. 2], two changes need to be performed. First, we need to flag the *PaddyFarmer*, *PC* (i.e. procurement company), *RPC* (i.e. rice processing company), *WholeSaler*, *IndustrialClient* and the *Retailer* as participant types. Then, we need to make the distinction between the different types of attributes as discussed in Sect. 3. In this case, we will mark *overallQuality* as a derived OCL attribute. The EDG drawn with B-MERODE is presented in Fig. 2, in which elements in bold and italic highlight changes from the original EDG described using only MERODE constructs.

Object-Event Table. The OET that is proposed with B-MERODE is visually identical to the one proposed in the MERODE model [18, Sect. 2.2]. There is however a need to specify which are the attributes that are modified by modifying methods. For instance, *EV_UPDATE_TRANSP_INFO* would trigger a method on a *PCOwnership* business object, which would modify the *transportation* attribute of that object. For all the modifying methods, this information needs to be specified in B-MERODE.

Finite State Machines. The FSMs in B-MERODE are exactly the same as in MERODE.

Event-Participant Table. In the EPT shown in Table 1, a "X" in a cell means that the business event type can be triggered by the corresponding participant type. The business event types that are in the EPT are the ones found in the OET, and the participant types are the ones described in the EDG. In this EPT, one particular participant type has been added: the collaboration manager (*CollabManager*). The reason why it was added is that organizations collaborating together in a network (such as a supply chain) need to define the rules according to which new participants can join the collaboration. As this is not discussed in detail in the present paper, a simplified way of handing that is to add a dedicated participant type, the only task and permission of it is to create instances of the different participant types. In a concrete case, the collaboration manager could take different forms. It could be an actual organization (acting then as a trusted third party), or it could be a smart contract containing the rules according to which participants can be added or removed.

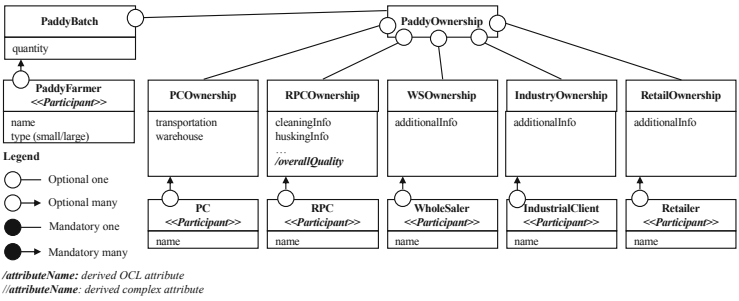

Fig. 2. Rice supply chain – EDG with B-MERODE

In the proposed model, every type of ownership (e.g. *PCOwnership* and *RPCOwnership*) is created by the participant type to which it corresponds (e.g. *PC* and *RPC*) when they acquire the batch. Therefore, only a procurement company (respectively, a rice processing company) can trigger the *EV_PC_TAKE_OWNERSHIP* (respectively *EV_RPC_TAKE_ OWNERSHIP*) event. Similarly, only the participant type related to a given type of ownership can mark it as finalized. Finally, the specific steps in the process that need to be executed by particular participant types and are recorded on the blockchain (e.g. cleaning step of the rice processing company) can only be recorded through the triggering of the appropriate business events (e.g. *EV_CLEAN_BATCH*) by the responsible participant type.

Attribute Reading Table. The second element of the permissions model that is described is the ART, available in Table 2. In that table, not all participant types are included (as BOTs from which attributes can be read). However, for all of them (except *PaddyFarmer*), the only attribute is *name* and it can be read by any participant type. For the *RPCOwnership* BOT, the scores related to the different processing steps (*cleaningInfo*, *huskingInfo*, etc.) are only readable by rice processing companies. However, the

overall score derived from these individual scores is available to all the participants upward in the supply chain (i.e. up to the retailers and industrial clients). The remainder of the table is self-explanatory.

Instance Access Rules. The last element to specify in the permissions model is the set of instance access rules (IARs). An example of IAR that would apply in the rice supply chain case is the following:

```
rule OnlyOwnRPCOwnership {
   operation: [ READ, UPDATE, DELETE ],
   businessObjectType(bo): RPCOwnership,
   participantType(ptcp): RPC,
   condition: bo.creator.id != ptcp.id,
   action: DENY
}
```

The rule states that to access a *RPCOwnership* BO (reading it, updating it or deleting it), a rice processing company must be the one that created the RPCOwnership.

(Core) Information System Services and Business Process Layer. To provide a complete picture of the system that would be used to support the rice supply chain using B-MERODE, there are 3 layers that need to be further discussed: the CISS, ISS and BP layers.

Table 1. Rice supply chain – EPT

	ParticipantType						
Business Event Type	PaddyFarmer	PC	RPC	WholeSaler	IndustrialClient	Retailer	*CollabManager*
EV_CR_PADDY_FARMER							X
EV_CR_PADDY_BATCH	X						
EV_CR_PADDY_OWNERSHIP	X						
EV_CR_PC							X
EV_CR_...	...	...	...	...	...	...	...
EV_PC_TAKE_OWNERSHIP		X					
EV_PC_FINALIZE		X					
EV_RPC_TAKE_OWNERSHIP			X				
EV_RPC_FINALIZE			X				
EV_..._TAKE_OWNERSHIP	...	...	...	...	...	...	...
EV_..._FINALIZE	...	...	...	...	...	...	...
EV_UPDATE_TRANSP_INFO		X					
EV_CLEAN_BATCH			X				
...	...	...	...	...	...	...	...
EV_IND_FINALIZE_ORDER					X		
EV_RETAIL_FINALIZE_ORDER						X	

Table 2. Rice supply chain – ART

	ParticipantType					
Attributes	PaddyFarmer	PC	RPC	WholeSaler	IndustrialClient	Retailer
PaddyBatch						
quantity	✓	✓	✓	✓	✓	✓
PaddyFarmer						
name	✓	✓	✓	✓	✓	✓
type	✓	✓	×	×	×	×
PC						
name	✓	✓	✓	✓	✓	✓
Retailer						
name	✓	✓	✓	✓	✓	✓
PCOwnership						
transportation	✓	✓	×	×	×	×
warehouse	✓	✓	×	×	×	×
RPCOwnership						
cleaningInfo	×	×	✓	×	×	×
...	×	×	✓	×	×	×
/overallQuality	×	×	✓	✓	✓	✓

In the CISSs layer, default services (as described in Sect. 3) are implemented, and no additional service needs to be defined as there are no derived complex attribute in the model. The ISSs layer describes a number of services providing reading and writing facilities for the different participants. For example, a service can be defined to read all the (accessible) information about paddy batches being processed by a given rice processing

company. Such services can be developed by the different participants as they wish to do so. Finally, in the BP layer, involved participants describe the internal BPs they implement to fulfill their duties as part of the collaboration. Just as ISSs, these processes remain at the liberty and discretion of the different parties involved.

5 Limitations and Further Research

In the present paper, we introduced B-MERODE, a novel MDE approach to generate BBISs supporting cross-organizational BPs. B-MERODE adopts the artifact-centric BP paradigm. Unlike other comparable methods, B-MERODE has the potential to offer increased flexibility and reusability, which we identified as important requirements. Furthermore, B-MERODE can further leverage the automation potential offered by MDE. After presenting our approach, we proposed an early validation by modeling a rice supply chain.

B-MERODE is still evolving and a number of further improvements can be made. In the current proposal, B-MERODE uses six distinct models as a consequence of the layered approach, which inevitably separates a number of concerns instead of representing them into a smaller set of models. Although the separation of concerns is crucial, in a next iteration we will study whether the number of distinct models could be reduced.

Another limitation relates to the practical application of B-MERODE. Although we conceptually motivate the approach and propose an early validation by means of a practical example, a larger-scale application would be an essential next step to strengthen the validation and would allow discovering some potentially remaining weaknesses in our contribution.

Further work should also consider architectural issues related to how existing information systems would, in practice, interact with BBISs generated using B-MERODE. Those architectural aspects were beyond the scope of this paper.

Most importantly, future work will also address the transformation step required to go from the proposed models to executable code. One of the benefits of the proposed approach is that it is entirely platform independent and that it can benefit from the existing expertise on MERODE transformations. Implementations of B-MERODE could generate BBISs that can be executed on any blockchain implementation, provided that the right transformations are available. Considering the investment in time and effort that would be required for people to be able to use B-MERODE, having the ability to test the developed models onto multiple platforms could increase the return on that investment.

With B-MERODE, the present paper aims at fostering adoption of blockchain technology. An important asset is its ability to generate working software from models in a single click. Extending this to blockchain will allow creating prototypes faster and at a lower cost (as explained in [6]). Facilitating the prototyping is crucial for organizations to successfully identify and capture strategic value from blockchain technology [21]. Furthermore, prototyping is a way to improve the outcome of an organization's learning process. Such an outcome is part of the drivers of the innovation of organization's business models [22].

References

1. Mendling, J., et al.: Blockchains for business process management - challenges and opportunities. ACM Trans. Manag. Inf. Syst. **9**(1), 1–16 (2018)
2. Seebacher, S., Schüritz, R.: Blockchain technology as an enabler of service systems: a structured literature review. In: Za, S., Drăgoicea, M., Cavallari, M. (eds.) IESS 2017. LNBIP, vol. 279, pp. 12–23. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56925-3_2
3. Amaral de Sousa, V., Burnay, C.: Towards an integrated methodology for the development of blockchain-based solutions supporting cross-organizational processes. In: IEEE 13th RCIS International Conference (2019)
4. López-Pintado, O., García-Bañuelos, L., Dumas, M., Weber, I.: Caterpillar: a blockchain-based business process management system. In: CEUR Workshop (2017)
5. Mavridou, A., Laszka, A.: Designing secure ethereum smart contracts: a finite state machine based approach. arXiv preprint arXiv:1711.09327 (2017)
6. Hull, R., Batra, V.S., Chen, Y.-M., Deutsch, A., Heath III, F.F.T., Vianu, V.: Towards a shared ledger business collaboration language based on data-aware processes. In: Sheng, Q.Z., Stroulia, E., Tata, S., Bhiri, S. (eds.) ICSOC 2016. LNCS, vol. 9936, pp. 18–36. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46295-0_2
7. Prybila, C., Schulte, S., Hochreiner, C., Weber, I.: Runtime verification for business processes utilizing the Bitcoin blockchain. FGCS **107**, 816–831 (2017)
8. van der Aalst, W.M.P.: Business process management: a comprehensive survey. ISRN Softw. Eng. **2013**, 1–37 (2013)
9. Reichert, M., Weber, B.: Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30409-5
10. Hull, R.: Artifact-centric business process models: brief survey of research results and challenges. In: Meersman, R., Tari, Z. (eds.) OTM 2008. LNCS, vol. 5332, pp. 1152–1163. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-88873-4_17
11. Chiao, C.M., Künzle, V., Reichert, M.: Integrated modeling of process- and data-centric software systems with PHILharmonicFlows. In: Workshop on CPSM (2013)
12. Tran, A.B., Lu, Q., Weber, I.: Lorikeet: a model-driven engineering tool for blockchain-based business process execution and asset management. In: CEUR Workshop (2018)
13. Madsen, M.F., Gaub, M., Høgnason, T., Kirkbro, M.E., Slaats, T., Debois, S.: Collaboration among adversaries: distributed workflow execution on a blockchain. In: Symposium on Foundations and Applications of Blockchain (2018)
14. Snoeck, M., Michiels, C., Dedene, G.: Consistency by construction: the case of MERODE. In: Jeusfeld, M.A., Pastor, Ó. (eds.) ER 2003. LNCS, vol. 2814, pp. 105–117. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-39597-3_11
15. Snoeck, M.: Enterprise Information Systems Engineering. TEES. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10145-3
16. Dedene, G., Snoeck, M.: Formal deadlock elimination in an object oriented conceptual schema. Data Knowl. Eng. **15**, 1–30 (1995)
17. Estañol, M., Queralt, A., Sancho, M.R., Teniente, E.: Artifact-centric business process models in UML. In: La Rosa, M., Soffer, P. (eds.) BPM 2012. LNBIP, vol. 132, pp. 292–303. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36285-9_34
18. Amaral de Sousa, V., Burnay, C., Snoeck, M.: B-MERODE: Application to a Rice Supply Chain (2019). http://bit.ly/2rGelZR. Accessed 29 Nov 2019
19. Xu, X., et al.: The blockchain as a software connector. In: Proceedings - 2016 13th Working IEEE/IFIP Conference on Software Architecture, WICSA 2016 (2016)

20. Kumar, M.V., Iyengar, N.C.S.N.: A framework for blockchain technology in rice supply chain management plantation. In: Future Generation Communication and Networking, pp. 125–130 (2017)
21. Plansky, J., O'Donnell, T., Richards, K.: A strategist's guide to blockchain. PwC report (2016)
22. Nowiński, W., Kozma, M.: How can blockchain technology disrupt the existing business models? Entrep. Bus. Econ. Rev. **5**, 173–188 (2017)
23. Ruiz, J., Sedrakyan, G., Snoeck, M.: Generating user interface from conceptual, presentation and user models with JMermaid in a learning approach. ACM International Conference Proceeding Series (2015)