Refining Case Models Using Cardinality Constraints

Stephan Haarmann[1(✉)], Marco Montali[2], and Mathias Weske[1]

[1] Hasso Plattner Institute, University of Potsdam, Potsdam, Germany
{stephan.haarmann,mathias.weske}@hpi.de
[2] Free University of Bozen-Bolzano, Bolzano, Italy
montali@inf.unibz.it

Abstract. Traditionally, business process management focuses on structured, imperative processes. With the increasing importance of knowledge work, semi-structured processes are entering center stage. Existing approaches to modeling knowledge-intensive processes use data objects but fail to sufficiently take into account data object cardinalities. Hence, they cannot guarantee that cardinality constraints are respected, nor use such constraints to handle concurrency and multiple activity instances during execution. This paper extends an existing case management approach with data object associations and cardinality constraints. The results facilitate a refined data access semantics, lower and upper bounds for process activities, and synchronized processing of multiple data objects. The execution semantics is formally specified by colored Petri nets. The effectiveness of the approach is shown by a compiler translating case models to colored Petri nets and by a dedicated process execution engine.

Keywords: Case management · Execution semantics · Petri nets

1 Introduction

Organizations apply Business Process Management (BPM) to specify, organize, analyze, and enact business processes. Models play an important role in documenting, improving, configuring, and monitoring these processes. Control flow-oriented process modeling languages, such as BPMN [22], are suited for well-structured processes. However, they lack support for semi-structured ones often required for knowledge work [5]. When executing knowledge-intensive processes, knowledge workers make informed decisions choosing from a set of possible continuations for a process. Thereby, they consult and maintain data objects.

Effective models of knowledge-intensive processes must capture flexible and data-centric behavior concisely and comprehensibly. Still, knowledge workers must decide how to progress each case in an ad-hoc manner. To address these requirements, among others, declarative [1,14,24] and artifact-centric modeling approaches [7,16,18] have been proposed. Declarative approaches concisely define

M. La Rosa et al. (Eds.): CAiSE 2021, LNCS 12751, pp. 296–310, 2021.
https://doi.org/10.1007/978-3-030-79382-1_18

processes with many variants through rules that eliminate undesired behavior. Artifact-centric models decompose processes based on data objects, called artifacts, or states thereof and combine the parts dynamically at run-time.

There also exist hybrid approaches such as fragment-based Case Management (fCM) [13,19]. fCM combines traditional BPMN-like control flow with a stronger focus on data objects. Processes are defined through multiple activity-centric process fragments. Each case manages a set of data objects, which can be accessed by all fragments of a given case. Consequently, the data requirements for process activities can lead to sophisticated dependencies among fragments. Data objects and their states are not only used for determining the process execution semantics but also for defining the goal of a case.

While previous works on fCM have focused on the approach from a conceptual perspective, process execution semantics have only been investigated informally. Behavior is specified in fragments, which are loosely coupled through data requirements and shared objects. Notably, these dependencies are not merely based on data objects and their states but also by their mutual associations and their cardinalities. In this paper, we focus on the following research question:

RQ: How can we formally characterize the execution of fCM models while considering data associations and cardinality constraints?

We present a thorough execution semantics for fCM case models, which takes into account process fragments and data objects with their respective states. Cardinality constraints of data objects are captured in the domain model and considered in activities' enablement conditions. The formalization is expressed as a translational semantics mapping fCM case models to colored Petri nets (CPN). It allows us to describe the process behavior precisely, to verify models formally, and to provide IT support during execution. In this regard, we present prototypes of a compiler (fCM to CPN) and a dedicated execution engine.

In Sect. 2, we present the fCM language by example. We specify the execution semantics in Sect. 3. We present our prototypes and a discussion in Sect. 4. Section 5 discusses related work. We conclude our paper in Sect. 6.

2 FCM Syntax and Notation

An fCM case model consists of a domain model, an object life cycle for each class in the domain model, fragments, and termination conditions [13]. We extend case models with data associations and cardinality constraints and thus get a full-fledged but non-hierarchical data model to represent structural aspects.

There is one domain model with associations and cardinality constraints for each case model. In practice, there may be only one domain model for the enterprise. However, the case class and goal cardinality constraints need to be defined for each case model. Domain model are visualized as UML class diagrams. We distinguish two types of cardinality constraints: *global cardinality constraints* must always hold, while *goal cardinality constraints* must hold when the case is closed. This accounts for "transient" states on the way towards the final state and is in the same spirit of what is supported by alternative approaches (cf. [1]).

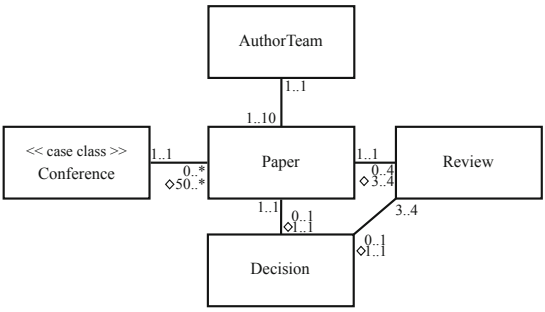

Fig. 1. The domain model of the submission and reviewing phase of an academic conference. Goal cardinality constraints have a leading ◇. They are depicted if and only if they refine the corresponding global constraints.

Consider a process for submitting and reviewing conference papers. The domain model (Fig. 1) contains classes for the conference, papers, authors, reviews, and decisions as well as associations among them. The conference is the case object and instantiated exactly once in each case. The association between the classes Paper and Conference together with its corresponding cardinality constraints requires exactly one conference for each paper and allows arbitrary many papers for each conference (global cardinality constraints). The goal cardinality must be at least 50, even though a conference exists independently of any paper.

For each class in the domain model, the case model has one object life cycle (OLC). An OLC contains a finite set of abstract states and transitions. fCM does not rely on initial and final states as other approaches do. In fact, the knowledge workers decide how to reach the case goal within the constraints given by the case model. A conference object, for example, can be in one of the states scheduled, open for submissions, closed for submissions, and reviewing closed (cf. Fig. 2). OLCs are not always sequences. They can contain disconnected parts in case of alternative initial states and branches in case of alternative state progressions.

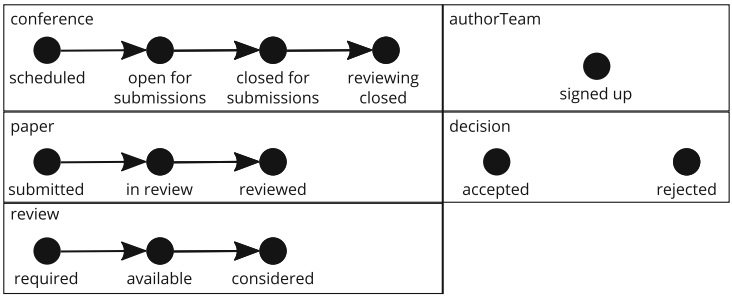

Fig. 2. Object life cycles for each class in the conference example (cf. Fig. 1) depicted as graphs. Each object life cycle consists of a set of states and state transitions between these states. States are represented by labeled circles, transitions by unlabeled arcs.

Objects are created and updated by activities. fCM places activities in acyclic control flow graphs called fragments (Fig. 3), which may additionally include XOR-gateways and start events. Activities read and write data objects, which are partitioned into input- and output sets, respectively. Furthermore, activities can operate on sets of objects[1]. All prerequisites fulfilled, fragments can run repeatedly and concurrently. While our example includes no XOR-gateways and fCM does not support loops and AND-gateways, decisions, concurrency, and loops can be modeled using alternative, concurrent, and repeatable fragments, respectively. In the example shown in Fig. 3, fragments *fb*–*ff* can be executed multiple times. Furthermore, instances of fragments *fc*–*ff* can run concurrently.

A case can end if a termination condition is satisfied (a goal is accomplished). For convenience, we define predicates called *object configurations* that require an object of a class in a state, e.g., *conference*[*scheduled*]. We call a set of object configurations a *data condition*. A termination condition is a data condition. The example has a single termination condition {*conference*[*reviewing closed*]}.

For fCM's formal definition, we refer to [13] and to our technical report [11].

3 Execution Semantics

A formal semantics is crucial to understand how fragments and objects interact, and for process analysis, execution, and monitoring. We present a semantics for fCM case models with data objects, associations, and cardinality constraints. We describe the execution of the example to provide an intuitive understanding. Then, we define a translational semantics targeting colored Petri nets (CPNs).

3.1 Walk-Through

A new case of the example in Sect. 2 begins once a conference is scheduled (*fa*). The start event creates the case object (conference). In general, non-initial fragments can begin after the start event; in the example, however, no fragment can be instantiated yet because the data prerequisites of their respective first activities are not satisfied. Thus, the conference must be opened for submissions first (*fa*) to fulfill the requirements of "submit paper" (*fb*).

When an author team submits their first paper (input set •, Fig. 3), a paper object and an author team object are created. Both get associated since objects that are created together and whose classes are associated get associated. Papers also get associated with the conference object as objects read get associated with newly created objects if their classes are associated. Global cardinality constraints must hold, i.e., an author team may submit not more than ten papers. Papers can be submitted concurrently (multiple instances of *fb*), but an instance of *fb* handles the submission and notification of only one paper. The submission is closed eventually (*fa*). Since no new papers can be accepted anymore, the conference-to-paper associations must meet the goal cardinality constraint. So, "close submission" requires 50 or more papers and updates them all.

[1] Previous fCM versions do not support set objects.

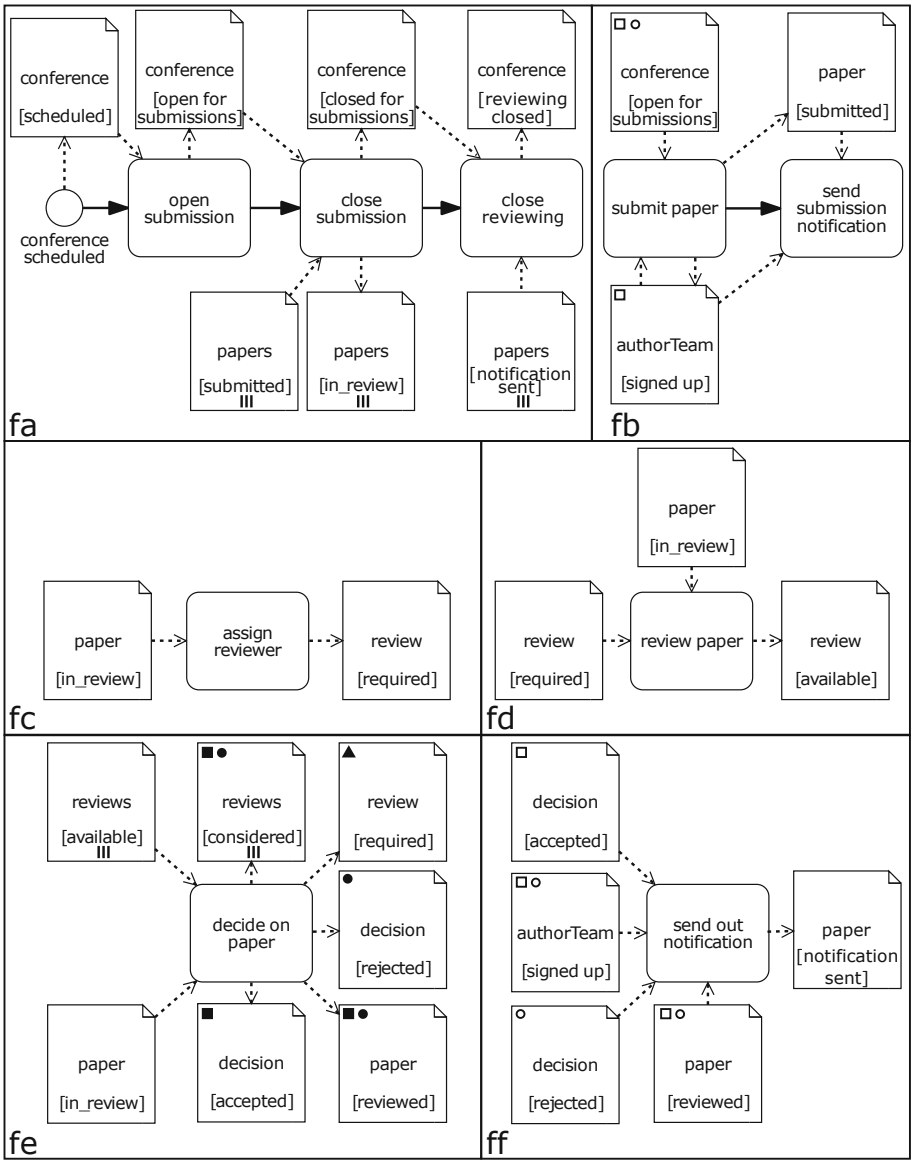

Fig. 3. Process fragments for the conference example. Each fragment is depicted as a graph consisting of BPMN elements. If an activity has more than one input- or output set, we added markers to the data object nodes to indicate the sets (in BPMN these are defined as attributes), e.g., submit paper has the input sets $\circ = \{conference[open\ for\ submissions]\}$ and $\square = \{conference[open\ for\ submissions], authorTeam[signed\_up]\}$ differentiating between an author team's first and any consecutive submission. This is visually indicated by the markers $\circ$ and $\square$, respectively.

After the submission has been closed, reviews for each paper are assigned (*fc*) and consequently created (*fd*). At most four reviews can be assigned to each paper (global cardinality constraint). Once all reviews for a given paper have been created, fragment *fe* can be executed: if there are less than three reviews, the outcome is an additional required review (output set ▲); if there are three reviews, the knowledge workers may decide that an additional review is needed or accept/reject the paper (output sets • and ■, respectively); if there are already four reviews, the first option (▲) is no longer applicable. Once a decision for the paper has been created, a notification is sent to the authors (*ff*). Once notifications for all papers have been sent, the reviewing phase is closed (*fa*). The whole case can be closed when the termination condition *conference*[*reviewing closed*] is fulfilled and all goal cardinality constraints are satisfied.

3.2 A Translational Semantics

As we have shown, the behavior defined in the fragments is significantly refined by the associations and their cardinality constraints. We define a translational semantics for fCM models to CPNs describing the behavior formally.

Assumptions & Preprocessing. To focus on the main aspects of the formalization and to avoid introducing additional complexity, we only support binary associations and at most one association between a pair of classes. This guarantees that, at the fragment level, no ambiguity arises when one wants to create a link between a pair of objects. Also, we assume that all the associations are existential. An association is existential if at least one of the corresponding global cardinality constraints has a positive lower bound. This means that an object of one of the involved classes cannot exist without an object of the other class. We call these objects *dependent* and *supporter*, respectively (in [26] such objects are called *slave* and *master*). Finally, we do not allow many-to-many associations.

To meet these condition, one has to operate on the data model via reification. A violating association is replaced by classes and multiple (compliant) associations. This comes at the price of enriching the vocabulary of the domain as extra classes and associations are added. For example, a many-to-many association A between classes c_1 and c_2 can be reified into a new class c_A related to c_1 and c_2 via two many-to-one associations. In some cases, such as reflexive associations, reification must be performed iteratively.

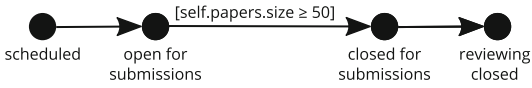

Fig. 4. OLC of the conference augmented with guards. The conference can only change to the state "closed for submissions" if there are at least 50 associated papers. This requirement arises from the goal cardinality constraints and the fragments: eventually, there must be 50 papers, and the fragments do not support adding papers for a conference once it is in the state "closed for submissions".

An activity that creates a *dependent* must read or co-create a *supporter*, i.e., "submit paper" must read the conference to create a paper. Such activities make assumptions about the states of the *supporter* object (i.e., *conference*[*open for submissions*]). If the state of a *supporter* is irreversibly changed to another, no further *dependents* can be created (i.e., *conference*[*closed for submissions*]); thus, the goal cardinality constraint must hold. This knowledge is encoded in the OLCs and the fragments: fragments describe the data requirements for each activity; OLCs describe valid state transitions, e.g., the OLC for the conference has no path from *closed for submissions* to *open for submissions*. Thus, we augment state transitions with guards to assert goal cardinality constraints (see Fig. 4).

Translation. We define the formal execution semantics for fCM by mapping case models to CPNs. A CPN is a Petri net with typed places and tokens. Transitions can have guards based on tokens and may derive new tokens from existing ones.

We define types, also called *colorsets*, to represent data objects, associations, and control flow. Objects are of type `ID`, where an ID consists of a class and an integer (i.e., number of objects of the same class at the point of creation). An association is an unordered pair of IDs. Instead of a token for each association, we use one token for the set of all associations. This representation for associations in CPNs is better suited than individual association tokens, since the content of a token can be queried. Also, places holding sets can be replaced by databases [21]. Additionally, we use a set of IDs as a registry for data objects. The control flow (`ControlFlow`) colorset has a field of type `ID` with the initial value `NULL` for each class. We also use tokens of the integer (`INT`) colorset and blank tokens (`Unit`).

The state of a case, including all data objects, associations, and control flow, is captured by tokens. We add places *i*, *r*, and *o* for the case's respective abstract states ready, running, and terminated (Fig. 5). The CPN contains a single place of colorset `Set<Association>` holding a token with all associations (Fig. 5) and one of colorset `Set<ID>` holding references to all objects (Fig. 5). We add a place for each object configuration (Fig. 6). An `ID` token on this place represents a data object. For each class, we add a place with colorset `INT` holding a token (with initial value 0) counting the number of respective objects (Fig. 6). Considering fragments, we add a `ControlFlow` place for each control flow (Fig. 7).

Events, activities, gateways, and termination conditions are represented by transitions. The firing of a start event moves a token from *i* to *r*, enables all non-initial fragments, and disables initial ones. Firing a termination condition

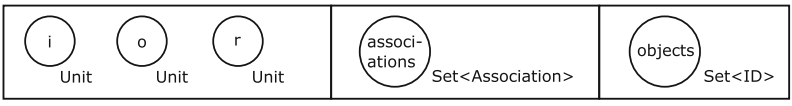

Fig. 5. Places that are created for each case model: an initial place *i*, which holds a single token until a case is started; a place *r*, which holds a single token while the case is running; and a final place *o*, which holds a token after the case has been closed. We add a place *associations* of type `Set<Association>` and a place of type `Set<ID>`.

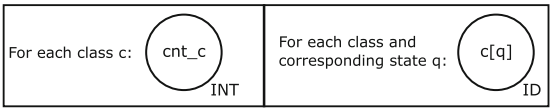

Fig. 6. To support data, two sets of places are added: a place with colorset `INT` for each class and a place of colorset `ID` for each object configuration (class and state).

moves a token from r to o. Transitions representing fragments' first activities consume and re-produce a token on r. If an activity has an incoming control flow, a corresponding `ControlFlow` token is required. A `ControlFlow` token is produced for outgoing control flow. The `ControlFlow` token tracks the data objects accessed by the fragment instance. So, this information is carried from activity to activity. After activity "submit paper" (*fb*), the `ControlFlow` token references the conference, author team, and paper. The next activity, i.e., "send submission notification", must not overwrite this information; thus, the same author team and paper are read. A CPN transition consumes (produces) tokens from (into) the same places every time it fires. However, fCM partitions inputs (outputs) of an activity into multiple alternative sets. To capture the I/O behavior of an activity, we create a transition for each combination of an input set with an output set (see Fig. 8). Such a transition consumes a token for each object configuration required by the input set and produces one for each object that is written or that remains unchanged. If a new object is created, the transition consumes and increments the corresponding counter. Its value is used to create the new object ID. If the created object (*dependent*) depends on another one (*supporter*), the set of associations is updated accordingly. Two potentially associated objects, e.g., author team and paper, can only be read together if they are associated (guard i in Fig. 8). Furthermore, an activity cannot be executed if it would violate the cardinality constraints (guard *ii, iii*, and *iv*).

Fragment *fb* (Fig. 3) has the activities "submit paper" and "send submission notification". The first has the input sets □ and ○ and one output set. It is mapped to two transitions, respectively (Fig. 9). Input set ○ captures the first submission of an author team: a token for the conference is consumed and reproduced, and new tokens for author team and paper are created. Associations between paper and conference and between author team and paper are established by updating the set of associations. Guard conditions assert the cardinality constraints. All three objects are referenced by the `ControlFlow` token. The transition for "send submission notification" reads the `ControlFlow` token and consumes the referenced paper and author team.

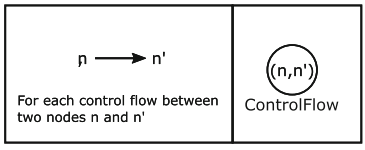

Fig. 7. For each control flow arc, a `Control-Flow` place is created.

Activities may process sets of objects. Assuming a set of type c_s in the state q_s is required, there must be an associated reference object in the input set. The CPN transition consumes the token for the reference object and one

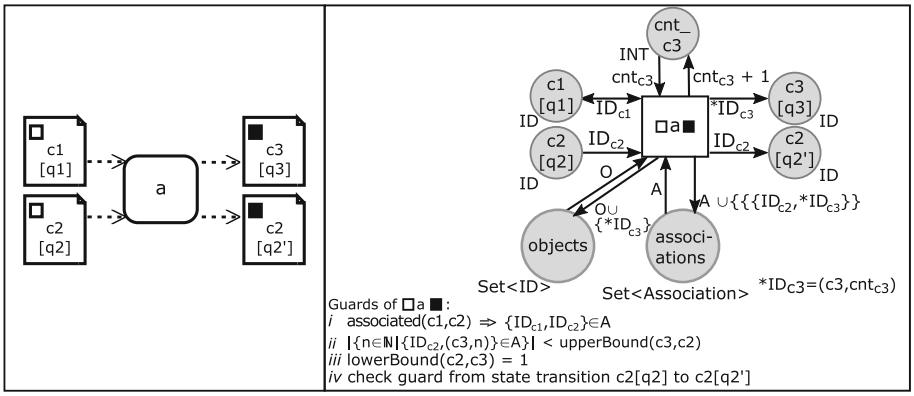

Fig. 8. Activity a with input set $\square = \{c1[q1], c2[q2]\}$ and output set $\blacksquare = \{c2[q2'], c3[q3]\}$. Objects can be read (c1,c2), updated (c2[$q2'$] with $q2 \neq q2'$), or created (c3). Associations (A) and the object registry (O) are read and updated. We assume an association between c3 and c2. The transition consumes and produces the required tokens. The guard asserts that (*i*) inputs are associated if applicable and cardinality constraints are met (*ii, iii*, and *iv*). For grey places see the mappings in Figs. 5 and 6.

for the associations to select the objects of class c_s that are associated with the reference object. If all these c_s objects are in the state q_s, the corresponding tokens are consumed. The output is similar to non-batch processing: read objects can be updated or remain unchanged. In any case, tokens for all read objects are produced. New objects may get associated with all objects in the set.

In fragment *fe* (Fig. 3), knowledge workers decide whether to accept a paper, reject it, or request an extra review based on the paper's available reviews. Each of the output sets (■, •, and ▲) is represented by a transition (Fig. 10). Since a decision requires at least three reviews (cf. Fig. 1), the two corresponding transitions (■ and •) can only fire if the set of reviews has three or four elements (upper bound). In the case of four reviews, the other transition (▲) is disabled.

We add a transition for each termination condition. It closes the case by moving the token from r to o and consumes tokens for each required predicate, the set of associations, and the set of objects to check goal cardinality constraints.

Due to space limitations, this paper does not include all details of the formalization. Interested readers are referred to our technical report [11].

4 Implementation and Discussion

The encoding of case models into CPNs detailed in Sect. 3 provides a basis for full-fledged enactment of fCM models. In this respect, we present prototypes of i) a compiler translating fCM models to CPNs compatible with CPNTools and of ii) an execution engine. Prototypes, documentation, and screencasts are available at https://github.com/bptlab/fcm2cpn/tree/caise and https://github.com/bptlab/fCM-Engine/tree/caise.

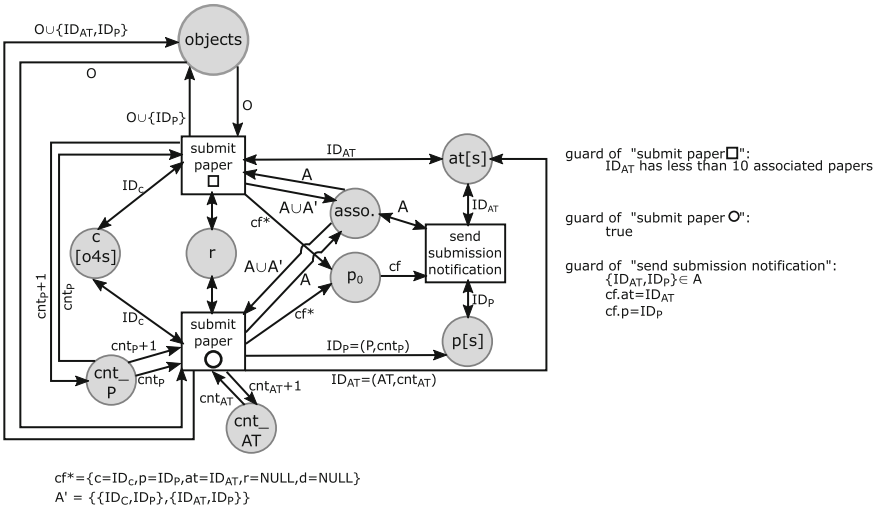

Fig. 9. CPN formalization of fragment *fb*. Types have been omitted for better readability. Places have been created by the mappings in Fig. 5 and Fig. 6. Transitions represent the activities and are connected to the places. Labels are abbreviated: c[o4s] = conference[open for submissions], at[s] = authorTeam[signed up], and p[s] = paper[submitted]. Initials of the classes are used for places and variables.

Case models can be created using common tools for BPMN and UML modeling. Given a BPMN file describing a set of fragments and a domain model (UML Class diagram without hierarchies) describing the classes of objects used in the case model, the compiler creates a CPN file using Access/CPN. We assume that the fragments conform to the OLCs, which, consequently, can be inferred.

The fCM engine takes the CPN and the corresponding domain model. The engine communicates with CPNtools to determine the enabled activity and event instances. The domain model is parsed to generate forms for data objects. Data objects' attributes and respective value assignments are managed outside of CPNtools. When a user selects an enabled activity, the engine displays the available input-output set combinations. The user can choose one, and the corresponding form is displayed (see Fig. 11). Once the user completes an action, the engine updates the objects and instructs CPNtools to execute the respective transition.

Table 1. Requirements of knowledge-intensive processes [5] fully (+) or partly (○) satisfied by fCM (+). We do not include requirements regarding user management and adaptability.

R1	Data Modeling	+
R4	Synchronized Data Access	○
R5	Data-driven Actions	+
R7	Formalized Rules and Constrains	○
R9	Goal Modeling	+
R11	Support for Different Modeling Styles	+
R12	Visibility of the Process Knowledge	+
R13	Flexible Process Execution	+
R24	Capture and Model External Events	+

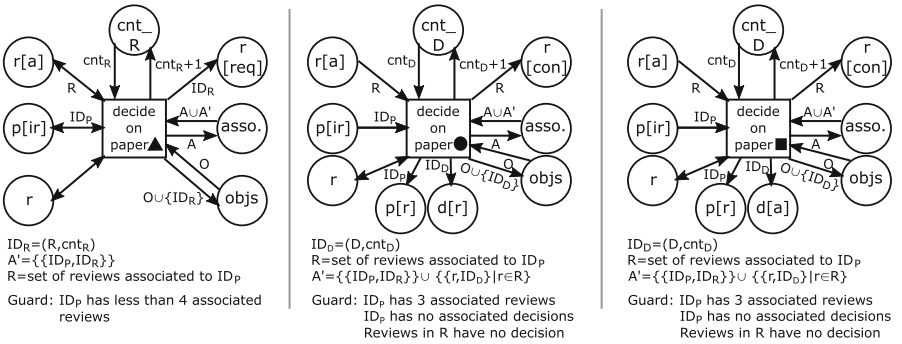

Fig. 10. Formalization of fragment *fe* split into three separated nets to ease comprehension. Types of the places are omitted and classes and states are abbreviated. The set of associations A is queried to derive the set of reviews R.

Table 2. Relevant marking for the CPN, see Fig. 10 for place names.

r	()
p[ir]	(Paper, 0)
r[a]	(Review, 0) (Review, 1)
asso	[((Paper, 0), (Review, 0)), ((Paper, 0), (Review, 1))]
objs	[(Paper, 0), (Review, 0), (Review, 1)]
countR	1

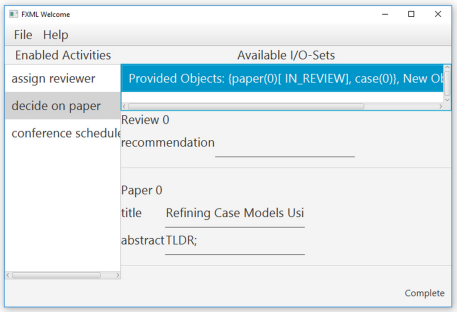

Fig. 11. The engine depicting the state from Table 2. A new review is created.

Tools are important to cope with the complexity of the approach and the domain. Our version of fCM contains hidden dependencies [10]: changing one part of a case model (e.g., cardinality constraints) may require adaptation of other parts (e.g., the inputs of activities). Additionally, the execution of fragments depends on the effects that other fragments induce over data objects and their associations. Well-designed tools for modeling, verification, and execution can compensate for some challenges [10]. The presented engine guides users and makes it optional to interpret the intricate models manually at run-time. Furthermore, fCM with its details may prove to be a proper low-code solution used by trained engineers rather than business users to design, implement, and monitor knowledge-intensive processes. However, this work focuses on the concepts and semantics. Notation and usability are left for future work.

We added domain models with associations and cardinality constraints to case models satisfying requirements for knowledge-intensive processes (cf. [5], Table 1): the process model integrates different perspectives each can be used according to their strengths. Furthermore, the process is data-driven—available

objects and cardinality constraints determine the enabled activities. Finally, data objects are captured in detail with states, associations, and constraints.

Data cardinality constraints are crucial for formal verification. In general, verifying models combining processes and data is undecidable [4], in particular, because the process may operate on unboundedly many data objects. As shown in [20], a data-aware version of soundness and other data-aware temporal properties can be verified if cardinality constraints with suitable upper bounds and adequately restructured queries are employed. This work takes a first step towards applying techniques studied in [20] and related approaches to the verification of fCM models, which is one of the next steps we want to take.

5 Related Work

We briefly present related work in three categories: fragment-based case management, alternative modeling approaches, and formal models/execution semantics.

fCM is a production case management approach introduced by Meyer et al. [19]. Hewelt and Weske describe fCM, focusing on the language and syntax [13]. Consecutive works propose methods for eliciting [12] and verifying [15] case models. These works do not consider associations or cardinality constraints. We add these to fCM, strengthening the link between data and behavior.

Data-centric process modeling approaches have a long history. Recently, Steinau et al. provided an overview of such approaches [28]. An early approach is MERODE, which interprets data models with existential associations as behavioral specifications [26]. In contrast to MERODE, fCM focuses on case management. Guard stage milestone [17] and CMMN [23] arrange activities in stages with data pre- and post-conditions. PHILharmonicFlows [18] models OLCs, which can be orchestrated into larger processes. Associations and cardinalities can be used to synchronize transitions between sets of objects [27]. BAUML [7] is an artifact-centric UML-based approach: class diagrams model artifacts, state machines the respective OLCs, activity diagrams refine the triggers of the OLCs, and OCL is used for constraints. BAUML is a generic *artifact first* approach. fCM on the other hand is more specific and tailored towards case management. In comparison to both BAUML and PHILharmonicFlows, artifacts are not the primary method of defining the process. Instead, activity-centric fragments define the process that creates, accesses, and alters data objects.

Rather than specifying process variants imperatively, declarative models define constraints to capture multi-variant processes concisely. Two prominent declarative languages are DECLARE [24] and DCR-Graphs [14]. Both use variants of linear temporal logics. While declarative modeling is powerful to model constraints from laws, guidelines, and other rules, imperative models are preferred when explicit control and data flow are important. DCR-KiPM [25] is a hybrid approach combining declarative DCR-Graphs with an imperative notion called KiPM to model knowledge-intensive processes. Object-centric behavioral constraints combine declarative process models with data models [1]. Activities and LTL rules over finite traces are quantified based on data objects and their

associations. Cardinality constraints exist in two flavors: they have to hold globally or eventually. However, no case exists. fCM achieves flexibility by combining fragments, modeled using elements within the BPMN tradition, dynamically. In fCM, declarative rules can be asserted through compliance checking [15].

Formal execution semantics are broadly applied in BPM. Dijkman et al. present a Petri net formalization for BPMN models enabling analysis and verification [6]. Proclets [2] are an extension of Petri nets: a process is composed of multiple workflow modules that contain synchronization points. They allow modeling various kinds of processes, such as choreographies and data-centric processes. Fahland introduces cardinality constraints for the synchronization points enabling many-to-many interactions [8]. DB-Nets assure that a process adheres to constraints given by a database (such as primary and foreign key constraints) by incorporating transactional semantics [21]. Catalog nets combine Petri nets with queryable read-only databases to support synchronization among cases [9]. Translating fCM to these formalisms would be interesting to connect the two trends of research. Object-centric Petri nets have been used to describe processes mined from object-centric event logs [3]. It would be interesting to see mining algorithms targeting fCM. Our semantics are defined using Petri nets, but users only interact with the fCM model, which hides the formalism's complexity.

6 Conclusion

Knowledge-intensive processes demand flexibility and data-driven actions. Also, modeling approaches should incorporate various perspectives through suited languages [5]. In this paper, we integrated activity-centric process fragments with domain models comprising associations and cardinality constraints. While we use fCM models, our work can benefit traditional processes (e.g., modeled using BPMN) to determine bounds for loops and multi-instance activities.

We showed that object cardinalities influence the behavior of processes. Activities are enabled depending on whether certain cardinality constraints hold. For example, a paper may only be rejected or accepted if three reviews exist (requirement). On the other hand, only one decision is made for each paper (bound). Furthermore, cardinality constraints may require batch processing, where multiple objects of the same type are accessed simultaneously. Additionally, cardinality constraints can refine the goal definition of a case. Not only do objects need to progress into specific states, but they also need to exist in specified quantities. This allows defining knowledge-intensive processes more comprehensively.

We provide a formal semantics using CPNs respecting the behavioral implications of cardinalities. The semantics is the underpinning for many BPM related tasks, such as process execution, verification, modeling, and mining. Due to the intricate nature of knowledge-intensive processes, tools are important. During modeling, a tool may highlight hidden dependencies and aid with the verification. At run-time, engines may guide the knowledge workers making it unnecessary to interpret the model manually. Alternatively, process monitoring may

check compliant actions. While we only focus on the definition of the formal semantics, we lay an important foundation for these future tasks.

We do not consider inheritance. Class hierarchies, where associations and cardinality constraints are the same on all levels, do not require any changes. However, specializing and generalizing associations including cardinality constraints and advanced modeling concepts, e.g., the phase pattern, are interesting for future work and require investigation of conceptual models and algorithms.

Future work may also investigate the role of knowledge workers in detail: they may require relaxed constraints or the ability to adapt the process ad hoc.

Acknowledgments. We thank Leon Bein for his work on the prototypes. Marco Montali acknowledges the UNIBZ CRC Project REKAP.

References

1. van der Aalst, W.M.P., Artale, A., Montali, M., Tritini, S.: Object-centric behavioral constraints: Integrating data and declarative process modelling. In: Proceedings of the 30th International Workshop on Description Logics, Montpellier, France, 18–21 July, 2017 (2017)
2. van der Aalst, W.M.P., Barthelmess, P., Ellis, C.A., Wainer, J.: Workflow modeling using proclets. In: Cooperative Information Systems, 7th International Conference, CoopIS, Eilat, Israel, 6–8 September, 2000, Proceedings, pp. 198–209 (2000)
3. van der Aalst, W.M.P., Berti, A.: Discovering object-centric Petri nets. Fundam. Informaticae **175**(1–4), 1–40 (2020)
4. Calvanese, D., De Giacomo, G., Montali, M.: Foundations of data aware process analysis: a database theory perspective. In: pods-13, pp. 1–12. ACM Press (2013)
5. Ciccio, C.D., Marrella, A., Russo, A.: Knowledge-intensive processes: characteristics, requirements and analysis of contemporary approaches. J. Data Semant. **4**(1), 29–57 (2015)
6. Dijkman, R.M., Dumas, M., Ouyang, C.: Semantics and analysis of business process models in BPMN. Inf. Softw. Technol. **50**(12), 1281–1294 (2008)
7. Estañol, M., Munoz-Gama, J., Carmona, J., Teniente, E.: Conformance checking in UML artifact-centric business process models. Softw. Syst. Model. **18**(4), 2531–2555 (2019)
8. Fahland, D.: Describing behavior of processes with many-to-many interactions. In: Application and Theory of Petri Nets and Concurrency - 40th International Conference, PETRI NETS, Aachen, Germany, June 23–28, 2019, Proceedings, pp. 3–24 (2019)
9. Ghilardi, S., Gianola, A., Montali, M., Rivkin, A.: Petri nets with parameterised data - modelling and verification. In: Business Process Management - 18th International Conference, BPM, Seville, Spain, September 13–18, 2020, Proceedings, pp. 55–74 (2020)
10. Green, T.R.: Cognitive dimensions of notations. People and computers V (1989)
11. Haarmann, S., Montali, M., Weske, M.: Technical report: Refining case models using cardinality constraints (2020)
12. Hewelt, M., Pufahl, L., Mandal, S., Wolff, F., Weske, M.: Toward a methodology for case modeling. Softw. Syst. Model. **19**(6), 1367–1393 (2020)

13. Hewelt, M., Weske, M.: A hybrid approach for flexible case modeling and execution. In: Business Process Management Forum - BPM Forum, Rio de Janeiro, Brazil, 18–22 September, 2016, Proceedings, pp. 38–54 (2016)
14. Hildebrandt, T.T., Mukkamala, R.R.: Declarative event-based workflow as distributed dynamic condition response graphs. In: Proceedings Third Workshop on Programming Language Approaches to Concurrency and communication-cEntric Software, PLACES, Paphos, Cyprus, 21st March 2010, pp. 59–73 (2010)
15. Holfter, A., Haarmann, S., Pufahl, L., Weske, M.: Checking compliance in data-driven case management. In: Business Process Management Workshops - BPM 2019 International Workshops, Vienna, Austria, 1–6 September, 2019, Revised Selected Papers, pp. 400–411 (2019)
16. 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
17. Hull, R., et al.: Introducing the guard-stage-milestone approach for specifying business entity lifecycles. In: Web Services and Formal Methods - 7th International Workshop, WS-FM, Hoboken, NJ, USA, 16–17 September, 2010. Revised Selected Papers, pp. 1–24 (2010)
18. Künzle, V., Reichert, M.: PHILharmonicFlows: towards a framework for object-aware process management. J. Softw. Maintenance Res. Pract. **23**(4), 205–244 (2011)
19. Meyer, A., Herzberg, N., Puhlmann, F., Weske, M.: Implementation framework for production case management: modeling and execution. In: 18th IEEE International Enterprise Distributed Object Computing Conference, EDOC, Ulm, Germany, 1–5 September, 2014, pp. 190–199 (2014)
20. Montali, M., Calvanese, D.: Soundness of data-aware, case-centric processes. Int. J. Softw. Tools Technol. Transfer **18**(5), 535–558 (2016)
21. Montali, M., Rivkin, A.: DB-Nets: on the marriage of colored Petri nets and relational databases. Trans. Petri Nets Other Model. Concurr. **12**, 91–118 (2017)
22. Object Management Group: Business Process Model and Notation (BPMN), January 2014. https://www.omg.org/spec/BPMN
23. Object Management Group: Case Management Model and Notation (CMMN), December 2016. https://www.omg.org/spec/CMMN
24. Pesic, M., Schonenberg, H., van der Aalst, W.M.P.: DECLARE: full support for loosely-structured processes. In: 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC), 15–19 October 2007, Annapolis, Maryland, USA, pp. 287–300 (2007)
25. Santoro, F.M., Slaats, T., Hildebrandt, T.T., Baião, F.A.: DCR-KiPN a hybrid modeling approach for knowledge-intensive processes. In: Conceptual Modeling - 38th International Conference, ER, Salvador, Brazil, 4–7 November, 2019, Proceedings, pp. 153–161 (2019)
26. Snoeck, M.: Enterprise Information Systems Engineering - The MERODE Approach. Springer, The Enterprise Engineering Series (2014)
27. Steinau, S., Andrews, K., Reichert, M.: The relational process structure. In: Advanced Information Systems Engineering - 30th International Conference, CAiSE, Tallinn, Estonia, 11–15 June, 2018, Proceedings, pp. 53–67 (2018)
28. Steinau, S., Marrella, A., Andrews, K., Leotta, F., Mecella, M., Reichert, M.: DALEC: a framework for the systematic evaluation of data-centric approaches to process management software. Softw. Syst. Model. **18**(4), 2679–2716 (2019)