

Case Model Landscapes: Towards an Improved Representation of Knowledge-Intensive Processes using the fCM-language

Fernanda Gonzalez-Lopez ·
Luise Pufahl · Jorge Munoz-Gama ·
Valeria Herskovic · Marcos Sepúlveda

Received: date / Accepted: date

Abstract Case Management is a paradigm to support knowledge-intensive processes. The different approaches developed for modeling these types of processes tend to result in scattered information due to the low abstraction level at which the inherently complex processes are represented. Thus, readability and understandability are more challenging than in imperative process models. This paper extends a case modeling language – the fragment-based Case Management (fCM) language – to a so-called fCM landscape (fCML) with the goal of modeling a single knowledge-intensive process from a higher abstraction level. Following the Design Science Research (DSR) methodology, we first define requirements for an fCML, and then review how literature – in the fields of process overviews and case management – could support them. Design decisions are formalized by specifying a syntax for an fCML and the transformation rules from a given fCM model. The proposal is empirically evaluated via a laboratory experiment. Quantitative results imply that interpreting an fCML requires less effort in terms of time – and is thus more efficient – than interpreting its equivalent fCM case model. Qualitative results provide indications on the factors affecting case model interpretation and guidelines for future work.

Keywords Case management · Knowledge-intensive process · Process landscape · Process map · Process architecture.

Fernanda Gonzalez-Lopez
Pontificia Universidad Católica Valparaíso, Valparaíso, Chile
Pontificia Universidad Católica de Chile, Santiago, Chile
E-mail: fgonlop@uc.cl

Luise Pufahl
Software & Business Engineering, Technische Universität Berlin, Germany

Jorge Munoz-Gama · Valeria Herskovic · Marcos Sepúlveda
Pontificia Universidad Católica de Chile, Santiago, Chile

1 Introduction

Case Management (CM) is a paradigm to support the design, execution, monitoring, and evaluation of knowledge-intensive processes (KiPs) [32]. Compared to procedural processes, KiPs are overall less structured. Instead, they are genuinely knowledge-, information-, and data-centric; and require substantial flexibility at design- and run-time [9,48]. KiPs are often found in domains where highly trained workers (i.e. *knowledge workers*) perform various interconnected knowledge-intensive decision-making tasks concerning very diverse units of work (i.e. *cases*). The term CM originated in the healthcare domain, where *medical personnel* – knowledge workers – deal with *patients* – cases – and the end-to-end process is not clear beforehand but is rather tailored on-the-go for each individual based on aspects such as examination results and medical team expertise [9].

In CM, a *case model* represents all possible courses of action for handling cases in a given KiP. As the CM approaches need to capture the complex behavior of KiPs – including processed data, possible operations on them, and their interrelations – case models tend to include more concepts and are more scattered than traditional imperative process models. When capturing flexibility, the routing and the control flow might be more difficult to understand compared to those of an imperative process model [27,53].

Different approaches and tools have been developed for CM, e.g. [20,24,40]. Though they are being used in the industry, e.g. [44], their adoption is still reduced. We argue that this is probably caused by understandability issues of the inherently complex case models. As posited by Davis [7], adoption in practice is correlated with ease of use. To address this design-time challenge, this work proposes accompanying a given case model with a complementary model, which we term *case model landscape* (CML). A case model landscape aims at representing a single KiP, emphasizing its sequence flow and data interdependencies. The goal of this article is to define how to create a CML for a case model in a specific CM approach: the *fragment-based Case Management* (fCM) approach [20]. Thus, the resulting CMLs are termed fCM landscapes (fCMLs). We focus on fCM because it is the most researched production case management approach with tool support; still, we will discuss how our proposal might concern other related approaches.

The present work followed the Design Science Research (DSR) methodology. This involves three main tasks: investigation, design, and validation [52]. We first identified the requirements for an fCML by extrapolating concepts found in *process overviews*, namely *process maps* [29,30], *process landscapes* [4,17], and *process architectures* [11,16]. Second, based on these requirements, we extended the modeling language used in fCM – namely the fCM-language – and defined the rules for generating an fCML from an existing fCM case model. Finally, we conducted a laboratory experiment to assess the interpretation of the resulting models by their readers.

The work reported in this paper is based on [15]. In comparison, we now provide a more structured description of fCMLs that includes formal descrip-

Table 1 Relevant terminology used throughout the paper.

Term	Definition
Case	Unit of work in the case management paradigm.
Case management (CM)	Paradigm for supporting knowledge-intensive processes.
Case model	Blueprint of a KiP in the context of case management.
Case model landscape (CML)	High-level view of a case model.
fCM case model	Blueprint of a KiP in the context of fCM.
fCM-language	Language for fCM case models based on BPMN.
fCM-language extension	Extension of the fCM-language proposed in this work, that allows modeling fCMLs.
fCM landscape (fCML)	High-level view of an fCM case model proposed in this work.
Fragment	Cohesive set of tasks and decisions constituting a structured part of a KiP.
Fragment-based CM (fCM)	A CM approach for unstructured processes with pre-defined segments.
Knowledge-intensive process (KiP)	Knowledge-, information-, and data-centric processes needing design- and run-time flexibility.
Process overview	High-level view of a collection of business processes.

tions for the fCM-language extension, and for the rules for building an fCML from a given fCM case model. Additionally, we offer a more elaborate empirical evaluation of the languages in a laboratory experiment, including both quantitative and qualitative data. This empirical assessment provided significant support for the preliminary findings reported in [15] stating that reading an fCML, compared to reading an fCM case model, requires less interpretation effort and improves interpretation efficiency.

In the remainder of the paper, foundations and related work are discussed in Sect. 2. Then, requirements for an fCML and our design decisions are presented in Sect. 3. The extension of the fCM-language for fCMLs is presented in Sect. 4 and its empirical evaluation is discussed in Sect. 5, followed by conclusions in Sect. 6. To ease reading the following, we provide a summary of some relevant terminology used throughout the paper in Table 1.

2 Foundations and Related Work

In this section, we present some grounding concepts of the paper and related work. Sect. 2.1 focuses on modeling languages and measuring model understandability. Sect. 2.2 discusses case management and alternatives to ease the case model understanding, and Sect. 2.3 introduces the fCM-language with the help of a meta-model and a running example. Finally, approaches for process overviews are presented in Sect. 2.4.

2.1 Modeling Languages and Understandability

A modeling language – language for short – contains the elements with which a model can be created and also interpreted. Modeling languages are described in terms of their abstract syntax, concrete syntax, and semantics [22]. The abstract syntax specifies the constituting concepts of the language and their

relations, while the semantics specifies the meaning of the language's concepts. The way to express a language is called concrete syntax or notation, which consists of a set of symbols and the rules to assemble them together coherently. The abstract syntax of a language can be represented using a meta-model. For example, the BPMN (Business Process Model and Notation) standard [39] provides meta-models that can be instantiated for building process models. The semantics of a language can be represented in different ways, e.g. BPMN uses mainly textual descriptions. However, formal mathematical descriptions are preferred for avoiding ambiguity. Finally, the notation of a language is represented by the mapping between the concepts of the language and a set of graphical symbols. For example, the concept of *event* found in BPMN is depicted as a circle. A notation is said to be cognitive effective if it allows generating models that can be easily, quickly, and accurately understood by their readers [36].

Overall, models are likely to be affected by the gulf of interpretation, i.e. the difference between what is understood by the reader of the model and what the modeler intended to communicate [38]. The extent to which this situation occurs for a given model is quantified in terms of interpretation performance, i.e. its effectiveness and efficiency. On one hand, *interpretation effectiveness* or simply *fidelity* accounts for how faithfully the interpretation of the model represents the original meaning of the model [5], e.g., the number of correct answers provided by a reader when using the model. On the other hand, *interpretation effort* accounts for the resources needed to interpret a model [5], e.g., the time needed by a reader to provide answers when using the model. *Interpretation efficiency* is the quotient between effectiveness and effort [5].

2.2 Case Management

Case management (CM) contributed to turning away from the strict activity view that had – so far – prevailed in business process management. A first approach for CM has been introduced as *Case Handling* in [1,3], which led to shifting the focus from activities to data. Similarly, *Business Artifacts* [37] with the Guard-Stage-Milestone (GSM) approach [24] focus on the high-level data – data artifacts – handled during case processing. A data artifact is defined by an information model and the allowed operations on them are expressed in a lifecycle model. GSM models consist of several stages reaching specific goals by the execution of defined operations; a stage can be entered if specific conditions are fulfilled. The GSM approach was used as the basis for the CMMN (Case Management Model and Notation) standard [40] which allows specifying case models showing, for example, optional and non-optional process fragments and their respective entry/exit conditions. However, some aspects of data – an essential element of CM – cannot be represented using CMMN, e.g. dynamic aspects of data such as their lifecycle states.

Despite an existing standard, other related approaches were continued or newly

developed, most prominently PHILharmonicFlows [26], fragment-based Case Management (fCM) [20], and declarative modeling approaches, such as [2, 42]. PHILharmonicFlows [26] splits a process into micro processes describing how a data artifact can be changed and a macro process handling micro processes relations. To deal with complexity, Steinau et al. [45] propose relational process structures representing the relationships between processes with cardinalities. However, aspects, such as the results exchanged by the process fragments or the enablement relations are not captured, limiting the understanding and the analysis of such a model. fCM by Hewelt and Weske [20] uses a small set of BPMN symbols for defining process fragments which can be combined at run-time according to data conditions. Additionally, the data structure and lifecycle states are provided for defining when the execution of a process fragment is enabled. In [19, 21], Hewelt et al. provide a method for supporting the case model elicitation. Still, it is an open challenge that the resulting case model is difficult to read for people not involved in the case model design. Declarative process modeling approaches, such as [2, 42], try to avoid the disadvantages of imperative process models by not specifying the process behavior a priori in a model. The activities of a declarative model can be executed in any order and with an arbitrary frequency. Only by adding rules to a declarative model (e.g. the execution of activity B always has to be preceded by activity A) the allowed behavior of a business process is constrained. However, experiments showed that declarative process models seem to be more difficult to comprehend [43]. A reason for lower understandability is that by different rules targeting the same activity implicit dependencies between activities can exist, which are not directly given in the declarative model. This is similar to the fCM model, where also the relations between the process fragments are also not explicitly shown in the model. De Smed et al. [8] propose an approach to detect automatically such implicit dependencies and visualize them in dependency diagrams. These types of diagrams have a quite low abstraction level which might lead still to understandability issues in the case of more complex models.

2.3 Fragment-based Case Management Language

As discussed in the introduction, our design decision is to develop a case model landscape for the fCM approach [20]. The fCM approach understands KiPs as having structured parts – i.e. *process fragments* – that are flexibly combined at run-time based on data handled by the process. This corresponds to the *unstructured processes with pre-defined segments* as defined by Di Ciccio et al. [9]: KiPs whose overall process logic is not explicitly defined, yet they entail pre-defined structured parts called fragments that need to be selected and properly composed on a per-case basis. Regarding its language, fCM reuses concepts from BPMN (Business Process Model and Notation) [39] – the widely applied industry-standard for imperative process modeling. We call this the fCM-language.

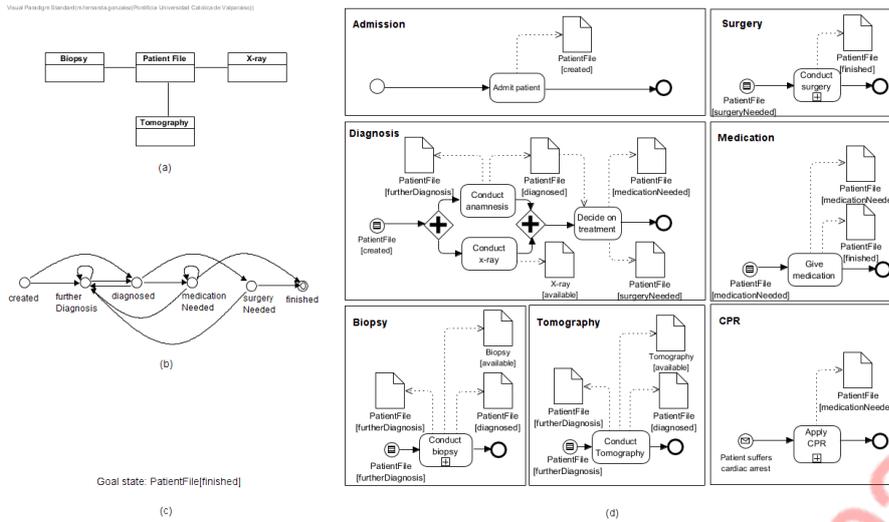


Fig. 1 Partial fCM case model for *medical consultation*: (a) domain model, (b) object lifecycle model for the *Patient File*, (c) goal state, and (d) process fragment models.

Each fragment of an fCM case model contains a number of activities and decisions, including consuming/producing data. Fragments are similar to subprocesses, with the difference that usually fragments have a condition that needs to be fulfilled before they can be executed. This is a data condition consisting of the existence of one or many instances of a given data type in a given state. Data and their states play a key role in the fCM approach. Besides being the element that allows combining fragments, a data condition is also used in fCM to express the termination condition of a KiP. In fCM, a case model consists of four sub-models: (a) a *domain model*, (b) a set of *object lifecycles*, (c) a *goal state*, and (d) a set of *process fragments*.

In the remainder of the paper, we use a running example termed the *medical consultation* process to illustrate the discussed concepts. In the example, when patients arrive at the hospital, they will be treated by a medical team for providing diagnosis and treatment and also by personnel for administrative matters, all with the goal of improving their health and discharging them from the hospital. The fCM case model for our running example is shown in Fig. 1. We use it in the following to illustrate the sub-models composing an fCM case model.

Domain model. The domain model represents the static view of the data that is relevant to the KiP. It is composed of a collection of data classes defining relevant data types. In the example in Fig. 1a, the relevant data types are *Biopsy*, *Patient File*, *X-ray*, and *Tomography*.

Object lifecycle models. Every data class of the domain model behaves according to a scenario-specific object lifecycle (OLC) or, simply, lifecycle. A

lifecycle depicts possible states and transitions that an object of a certain data type may undergo during the handling of a case. Fig. 1b shows the lifecycle of the `Patient file` as a finite state machine with the following possible states: *created*, *furtherDiagnosis*, *diagnosed*, *medicationNeeded*, *surgeryNeeded*, and *finished*.

Goal state. The goal state defines when an instance of a given case model can terminate. The goal state corresponds to a set of data conditions (i.e. availability of a data object of a given class in a given state) related via logical operators. Fig. 1c shows the goal state for our running example: a `Patient File` has to be in the state *finished*.

Process fragment models. A case model contains multiple process fragments. In the example, the fragments are *Admission*, *Diagnosis*, *Surgery*, *Medication*, *Biopsy*, *Tomography*, and *Cardiorespiratory Resuscitation (CPR)*, as depicted in Fig. 1d. Each process fragment is composed of a set of data, gateway, event, and activity nodes linked by edges, as in BPMN (Business Process Model and Notation) process models. In the remainder of the paper and as shown in Fig. 1d, data types and their states are depicted using the BPMN data-object notation: `Data type [state]`.

If the start event of a fragment is a blank start event, then the fragment is enabled with case initiation, e.g., the *Admission* fragment. Else, if the fragment has a message start event, it is enabled as soon as the external event occurs during case execution. External events can be messages by other cases, or relevant signals from the case environment, e.g., the patient suffers cardiac arrest as shown in the *CPR* fragment in Fig. 1d. Most fragments, however, are enabled by a data condition, which means that such fragments are only enabled to start once a given data condition is true. This is represented by a conditional start event referencing a certain data condition. For example, as shown in Fig. 1d, the *Diagnosis* fragment becomes enabled when there is a data instance of `Patient file [created]`.

Fragment modeling requires consistency in labeling data nodes and data conditions to capture the relations between fragments. The activities within the different fragments consume/produce data in given states. For example, in Fig. 1d the *Admission* fragment writes a `Patient file [created]`. It is possible to identify a relation between the *Admission* and the *Diagnosis* fragments, since data produced in the former, enables the execution of the latter.

In fCM, knowledge workers have a central role and can decide which of the *enabled* fragments are executed and how often based on the allowed data conditions. For instance, if the `Patient file` is in the state `[furtherDiagnosis]`, the fragments *Biopsy* and *Tomography* are enabled. The knowledge worker can then decide which of them to be executed. As soon as one of the two fragments is started, the other fragment is not enabled anymore because the data `Patient file [furtherDiagnosis]` is not available anymore [20]. Still, after executing one of them, the patient file can be again in the state `[furtherDiagnosis]`,

such that both fragments can be executed in one case. Formally, we define an fCM case model as follows:

Definition 1 (fCM case model) An fCM case model is defined by the tuple $cm = (name_{cm}, D, G, F)$ where:

- $name_{cm} : cm \rightarrow String$ is a function assigning a case model a label.
- $D = (C)$ is the domain model of the case where C is a non-empty finite set of classes. Let $c \in C$ be a data class, and Q_c the set of possible states for objects of class c . Furthermore, let $t \subseteq Q_c \times Q_c$ be the set of valid state transitions for the class c . The object life cycle of c is a state transition system $l(c) = (Q_c, t)$. $l(C)$ and describes the set of object life cycles corresponding to the classes in C .
- $G = \{cond_1, cond_2, \dots, cond_n\} \subseteq \mathfrak{P}(DC)$ is a set of goal conditions (being a subset of power set of DC). Each $cond_i \in G$ can consist of several data conditions, which need to be fulfilled. A data condition $dc \in DC$ is in form of $c[q_c]$, where $c \in C$ and $q_c \in l(c).Q_c$. The case can terminate if for any $cond_i \in G$, all data conditions $dc \in cond_i$ are satisfied.
- F is a non-empty, finite set of fragments.

Definition 2 (Fragment) Let $f \in F$ be a fragment defined by a tuple $f = (name_{fra}, N, \rightarrow, s, read, write)$ where:

- $name_{fra} : f \rightarrow String$ is a function assigning a fragment a label.
- $N = N_A \cup N_G \cup N_E$ is a non-empty, disjoint set of activities N_A , gateways N_G , and events N_E .
- $\rightarrow \subset (N) \times (N)$ is the control flow between the nodes.
- $s \in \{blank, conditional, message\} \subset N_E$ is the start of a fragment and the function $\sigma : s \rightarrow Cond = \{cond_1, cond_2, \dots, cond_n\} \subseteq \mathfrak{P}(DC)$ assigns the start a set of enablement conditions:
 - a fragment with $s = blank$ is enabled and can be started anytime by a knowledge worker,
 - a fragment with $s = conditional$ is enabled as soon as $cond_i \in \sigma(s)$ is available and all $dc \in cond_i$ are satisfied,
 - a fragment with $s = message$ is enabled if a certain external event occurs.
- $read : A \rightarrow \mathfrak{P}(DC)$ assigns each activity a set of data conditions. For an activity $a \in A$, $read(a)$ returns the data conditions that can be read by a . $READ : f \rightarrow \mathfrak{P}(DC)$ provides for a given fragment f a set of data conditions read by the activities N_A of the fragment f .
- $write : (A \cup \{s\}) \rightarrow \mathfrak{P}(DC)$ assigns each activity a second set of data conditions. Given an activity $a \in A$, $write(a)$ returns the data conditions that can be written by a . $WRITE : f \rightarrow \mathfrak{P}(DC)$ provides for a given fragment f a set of data conditions written by the activities N_A of f .

2.4 Process Overviews

Process overviews – a term used in this paper for referring either to a process map, landscape, or top-level model of a process architecture (often referred to simply as process architecture) – support reasoning and analysing the structure of a business process collection, leaving aside much detail of individual processes [10,13]. In such level of abstraction, individual processes are depicted as black boxes and, therefore, the focus of the model is on the structure of the process collection [10,13]. Compared to detailed process models, process overviews allow representing more straightforwardly: (a) high-level concepts regarding a single process, such as inputs/outputs; as well as (b) concepts regarding the relationships between processes, such as control and data flow. We briefly discuss different process overview approaches in the following.

Process maps are usually easily readable by non-technical users due to being modeled with a small set of concepts with lax semantics. It might consist solely of a hierarchical classification of processes, or also inputs/outputs of the constituting processes might be specified [31]. In the Process Map approach by Malinova [30], data is considered at a very high level of abstraction: data-flow between processes is shown as directed arcs while leaving out details about the involved data.

Process architectures are more technically oriented and each represented concept has precise semantics. An example of a language for process architectures is provided by Eid-Sabbagh et al. [11,12]. In this approach, although data is used for the identification of process relations, the language does not include elements for the explicit representation of data. Relations are represented by connecting events that are relevant to the processes. An interesting aspect of the approach by Eid-Sabbagh et al. is the use of logic operators in the architecture models: these operators show conjunction and exclusion rules for the execution of processes.

ArchiMate [47] has become the industry standard for modeling enterprise architectures and can be used to model process architectures as well [14]. The standard supports the representation of data, but not of their states.

Process landscapes could be seen as the middle ground between process maps and architectures. Proposals in this area also struggle with the issue of ensuring an adequate level of understandability, e.g. [4,17].

Altogether, multiple approaches have been proposed to convey overviews for collections of processes. The present work extrapolates and reuses some concepts of process overviews for generating a high-level overview of a given fCM case model. An analogy is made between processes in a process overview and fragments in an fCM case model. This analogy is the base for creating a new model, termed fCM landscape (fCML) that is complementary to the fCM case model from which it was generated.

3 Requirements and Design of a Case Model Landscape

As previously stated, this work focuses on a specific approach for case modeling named fragment-based Case Management (fCM). Our aim is proposing an extension for the fCM-language, i.e. the language used for case models in fCM. Such an extension will allow to create new views for fCM case models: fCM landscapes (fCMLs). fCMLs pursue capturing essential aspects of fCM case models while leaving aside some detail to provide better understandability of control flow and data aspects.

Requirements for an fCML are defined in Sect. 3.1 together with general design decisions. Finally, detailed design decisions are discussed in Sect. 3.2.

3.1 Requirements and general design decisions

Some works present potential requirements and classification frameworks for KiPs to be reused and extended, e.g. [9,46]. However, we believe it is more adequate to define the requirements for an fCML based on the limitations in the understandability of fCM case models. Particularly, previous studies like [19] show that readers find it challenging to understand some aspects of the control perspective (i.e. the logical order of fragments) and data perspectives (i.e. how information is exchanged between fragments) in an fCM case model. However, the control and data flow are key to the comprehension of processes at design-time [51]. We advocate that the understandability challenges are mainly rooted in two main features of fCM case models: namely, their low abstraction level and their use of sub-models. Having this in consideration, this work proposes using an fCML as a complementary model for an fCM case model that overcomes the aforementioned issues. The requirements for an fCML and our general design decisions to target these requirements are summarized in Table 2 and described in the following:

- **R1. An fCML should be consistent with the respective fCM case model.** An fCML provides an alternative view of a process represented by an fCM case model. Consequently, both models must be consistent in terms of representing the same process. This requirement involves that building an fCML from a given fCM case model should be possible by applying a set of mapping functions to the fCM case model to obtain the fCML.
Design decision: The proposed work defines such a set of mapping functions.
- **R2. An fCML should be a coarse grained (simplified) representation** of its corresponding fCM case model. The fine grained fCM case models imply a high cognitive load, which makes it harder to find the specific pieces of information in the model [19]. For example, a large number of event nodes in an fCM case model need to be visually inspected in order to find the start node.
Design decision: The abstraction level of the model is modified in our proposal by hiding the detailed logic of each process fragment. Accordingly,

process fragments in an fCML will be depicted as black-boxes, analogously to how processes are depicted in process overviews, e.g. [11,30].

- **R3. An fCML should consist of a single model.** While the use of multiple sub-models in fCM case models allows to decouple different types of process information, the reader of such model needs to inspect multiple sub-models to find some information, e.g. the reader needs to inspect every fragment model plus the goal state sub-model before they can identify the end of the KiP. A single model is needed to understand how the detailed models are in connection to each other similar to what process overviews provide for a collection of business processes [10] and the coordination process of Philharmonic flows for a set of object lifecycles [6].

Design decision: An fCML is proposed as a single model based on the diverse sub-models for a given fCM case model.

- **R4. An fCML should highlight the control perspective of the KiP.** Understanding the control flow between process fragments of an fCM case model can be challenging since the reader needs to inspect multiple nodes, namely data conditions and data outputs of each process fragment [19,53]. Thus, it is the aim to provide the reader a better understandable model of the fCM with the control flow connection from the start to the end.

Design decision: The proposed work highlights the control perspective extrapolating some concepts used in procedural models [39] and some process overview approaches such as [11,30].

- **R5. An fCML should highlight the data perspective of the KiP.** Data plays a key role in knowledge-intensive processes [9]: In fCM, process fragment models show data produced/consumed during the process. Data is essential for the enablement of fragments as well as describing the case goal, such that data exchanged between fragments should be captured in a case model.

Design decision: The present work wants to highlight the data perspective by showing data nodes as part of the process flow in fCMLs.

3.2 Detailed Design Decisions

Together with fCM [20], a set of languages for process overviews and CM approaches was assessed to identify concepts that could be extrapolated to fulfill the requirements for an fCML previously discussed in Sect. 3.1. The justification for selecting these works is that they are either the industry standards in their fields – ArchiMate [47] and CMMN (Case Management Model and Notation) [40] –, or they are representative and well-documented proposals from the research community – Process Architecture by Eid-Sabbagh [11,12] and Process Maps by Malinova [30].

- **High-level overview.** An fCML is required to map information scattered between the multiple sub-models of an fCM case model into a single model (see R1 in Sect. 3.1). We decide to enclose this high-level overview of the KiP in a container. This design decision is based on the fact that approaches

Table 2 Overview of requirements and design decisions.

Requirement	Design decisions
R1. An fCML should be consistent with the respective fCM case model	A set of mapping functions is defined to generate an fCML based on a given fCM case model.
R2. An fCML should be a coarse grained (simplified) representation of its corresponding fCM case model.	Each fragment in an fCM case model is collapsed into a single notational element with the name of the respective fragment in the fCML.
R3. An fCML should consist of a single model.	fCML as a labeled container enclosing a high-level overview from the start to the end of an fCM case model as graph-based representation.
R4. An fCML should highlight the control perspective of the KiP.	An fCML explicitly represents when a case can start/end using events. Between these points, control flow is depicted using directed connectors and logic operators.
R5. An fCML should highlight the data perspective of the KiP.	Data is explicitly represented in an fCML. Data nodes in an fCML show the data that is read/written by fragments and the goal state of the case model.

for CM and process overviews consider often a container specifying the limits of what lies within the case model (e.g. [40]) or process collection (e.g. [30]), respectively. Similarly, we want to present an fCML as a labeled container enclosing a high-level overview from the start to the end of an fCM case model as graph-based representation.

- **Case start.** A key aspect for understanding control flow in a process (see R4 in Sect. 3.1) is identifying when it starts. An fCM case model can be started by a knowledge worker at any time and then all fragments with a blank start event are enabled. Additionally, external events can trigger the execution of a new case. For an fCML, we want to represent the manual start of a KiP explicitly as a case start connecting to all enabled fragments, whereas external events will be covered by the external triggers later. The use of events was found to be the most common way to represent the start in the reviewed works (e.g. [11, 40, 47]).
- **Data object.** Being a data-centric approach, data plays a key role in fCM case models (see R5 in Sect. 3.1). In fCM, the term data object corresponds to a data type that holds one of its lifecycle-defined states¹. Data is explicitly represented in CMMN [40] and some process overviews approaches (e.g. [30, 47]). However, these works omit the specification of a lifecycle-defined state considered in fCM. We, therefore, decide that an fCML will maintain the representation of data types used in fCM case models: BPMN data objects labeled with the name of the data type and its state.
- **Data condition.** In fCM, data objects may constitute data conditions, which include the combination of data objects via logic operators. Data conditions are involved in both the data- and the control- perspective of fCM case models (see R4 and R5 in Sect. 3.1) because they determine

¹ In other contexts, e.g. UML, a data object corresponds to an instance of a data type.

data-based process fragment relations and the case end. Regarding data conditions, no other reviewed work considers them in the sense that they are considered in the fCM approach. CMMN [40] includes data conditions, but only to a limited extent, since there is no notation for such an element and, instead, it is considered as a parameter of another element called sentry. In an fCML we include the concept of data conditions as in fCM case models, but without using the same notation (i.e. BPMN conditional start event). Instead, we decide to combine the symbols for data objects and pre-requisites.

- **Case end.** In fCM, the case end is enabled by fulfilling the data condition termed the goal state. The goal state needs to be true before a knowledge worker can terminate the KiP. The goal state is not included in fragment models but in another sub-model, demanding that the reader reviews multiple pieces of information to understand this data-related aspect of control flow (see R3, R4, and R5 in Sect. 3.1). The ending point is represented in different ways among the reviewed works: as an event [11], an external output [30], or a goal [47]. None of these approaches is expressive enough to represent the fCM concept of enabling case end via reaching a goal state. Representation of case end in an fCML is rather inspired in BPMN. Case end is represented by a BPMN blank end event and it is linked to the respective data goal condition.
- **External trigger.** The fCM approach considers that external events may trigger a new case or certain process fragments in a running case. This feature is key to control flow (see R4 in Sect. 3.1). As in the fCM approach, all other reviewed works (see [11,31,40,47]) represent external triggers as events. Each of these approaches uses a different symbol for that. For fCMLs, we decide to reuse the symbol for external trigger in fCM case models: the BPMN message start event.
- **Fragment.** An fCM case model offers a detailed representation of each process fragment constituting the respective KiP. The granularity of this representation poses a challenge for the reader interested in having an overview of the control flow (see R2 and R4 in Sect. 3.1), and thus fCMLs should provide a more abstract representation of process fragments. In CMMN, the concept of fragment is embodied by the so-called stages [40]. The analogous concept for fragments in process overviews is that of a sub-process (see [11,31,47]). Based on this, we decide that in an fCML each process fragment model should be collapsed into a single notational element with the name of the respective fragment.
- **Fragment pre-requisite.** In the fCM approach, each fragment is said to have one pre-requisite that needs to be fulfilled before the fragment can be executed. This includes three possibilities: (i) *blank* corresponding to the lack of a pre-requisite and can manually be started any time (cf. case start); (ii) *external trigger* corresponding to a fragment that is enabled by an external event (cf. external trigger), and (iii) *data condition* corresponding to a fragment that can only be executed once some data condition is true. Being able to identify these possibilities is critical for understanding the

control flow of the KiP (see R4 in Sect. 3.1). The concept of pre-requisite is found in CMMN [40]. This language provides an element called entry criterion depicted as a blank diamond. In fCMLs, we plan to reuse this element at the border of the respective fragment and linked it to the respective pre-requisite (i.e., a BPMN event of type blank, external trigger, or data condition).

- **Fragment relations.** A key aspect of fCM is that the relations between fragments are based on data: when certain data is created by a fragment, then other fragments might be enabled, such that they can be executed by the knowledge worker. However, some effort is required for identifying these data-based relations in fCM case models (see R4 and R5 in Sect. 3.1). In the reviewed works, process/fragment relations are represented using - sometimes differentiated - connectors [11,31,40,47] and sometimes also logic operators [11,40,47]. For fCMLs, fragment relations we also use connectors and logic operators but, additionally and for the sake of expressiveness, the involved data objects should also be shown. The logic operators considered are AND, OR, and XOR. However, in our original proposal [15] we had merged the OR and XOR operators into one symbol. This constitutes a semiotic clarity problem [36] identified after the user study. We solved this by providing one symbol per semantic construct, i.e. XOR and OR.
- **Fragment optionality.** A central aspect of fCM is that fragments are combined depending on the case at hand. This means that only some fragments are included in all possible executions of a KiP. We rank the concept of fragment optionality as useful for improving the expressiveness of the control flow (see R4 in Sect. 3.1) since it eases the differentiation between the fragments that are always executed and those that are not. Only CMMN [40] defines an explicit notational element for showing which parts of the case model are optional. This work depicts discretionary elements as dashed-edged and we want to reuse them as well for fCMLs.

Table 3 Modeling elements of the fCM-language extension for fCML.

Element	Description	Notation
Case end	End of the model, it is enabled due to achieving the goal state.	
Case start	Start of a case model. If a new case of the model is instantiated then it is started with the succeeding process fragment.	
Connector	Causal relation between the elements of the model.	
Data object	Data type holding a particular state in which it is consumed/produced by a process fragment.	
Data object, goal state	Data condition that must be fulfilled for the termination. This can also be a combination of data conditions via logic operators.	
External trigger	External occurrence of an event, which is relevant for the case. It enables the start of the succeeding process fragment.	
fCM landscape	Container of a landscape for a specific case model.	
Logic operator, AND	Forking or merging of paths following the logic of a logical AND-operator.	
Logic operator, OR	Forking or merging of paths following the logic of a logical OR-operator whereby multiple fragments are enabled but only one can make changes on one data type at a time.	
Logic operator, XOR	Forking or merging of paths following the logic of a logical XOR-operator.	
Pre-requisite	Data pre-condition or an event that enables the start of a process fragment.	
Process fragment, non-optional	Non-optional process fragment that needs to be executed in every possible execution of the KiP.	
Process fragment, optional	Optional process fragment that is not necessarily executed in every possible execution of the KiP.	

4 Extension of fCM-language for Modeling Landscapes

After identifying its requirements and discussing design decisions in view of related works, this section provides a detailed description of the extension of the fragment-base Case Management (fCM) language for modeling an fCM landscape (fCML). In this section, we first present the language rather informally based on the running example of the *medical consultation* in Sect. 4.1, and then, we introduce the abstract syntax and the semantics of the elements of an fCML in Sect. 4.2.

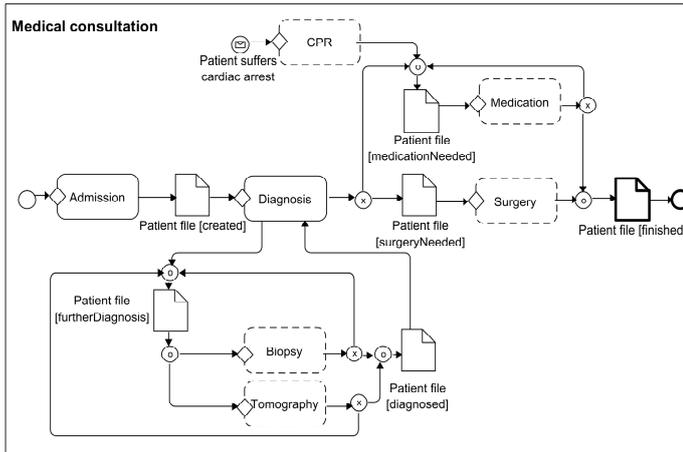


Fig. 2 Case Model Landscape for *medical consultation*.

4.1 Exemplified Case Model Landscape

Fig. 2 shows an fCML for the medical consultation knowledge-intensive process (KiP) example on which we will explain the elements of an fCML. It is generated from the fCM case model in Fig. 1. As an overview, the elements of the proposal, a short description, and the notation are given in Table 3.

The case model always starts with the *Admission* fragment, which follows the blank start event. Each fragment has a pre-requisite shown as an unfilled diamond at the border of the fragment. The pre-requisite describes the condition that must be satisfied to start a fragment. It is also possible that data is produced (written) or consumed (read) by a fragment. The *Admission* fragment has no data condition – it simply starts by initiating a new case – but it produces a `PatientFile[created]` data object, needed as a pre-requisite by the *Diagnose* fragment.

During the execution of the *Diagnose* fragment, a `PatientFile[furtherDiagnosis]` can be produced which is visualized by an outgoing arc from the fragment connected to the data object. If the data object is available, the optional *Biopsy* fragment or the optional *Tomography* fragment, or both can be executed in one case. Still, if one of the fragments is started, the other one cannot be executed and do changes on the data object which is currently used by the other fragment. This construct is represented by a logical OR-operator connected to the pre-requisite of both fragments. These two fragments do not need to be executed in every case, they are optional which is shown by a dotted borderline. Both fragments could produce a data object `PatientFile[diagnosed]` in *furtherDiagnosis* or *diagnosed*, also represented with the help of a logical OR-operator. In the case of *furtherDiagnosis*, the two just discussed fragments can be restarted. In the other case, the *Diagnose* fragment is continued, which is shown by the incoming connector into the fragment symbol.

This fragment produces either the `PatientFile` in *surgeryNeeded* or *medicationNeeded*, enabling the optional fragments *Surgery* or *Medication*, respectively. Both the fragments can produce `PatientFile[finished]` representing the goal state of the model and leading to the end event, the end of the model. The *Medication* fragment can also result in `PatientFile [medicationNeeded]` as alternative, re-enabling this fragment.

During the execution of the KiP, also a relevant event for this business scenario can occur – *Patient suffers from cardiac arrest*. Represented by a message start event, this event triggers the *CPR* fragment. It also results in the `PatientFile [medicationNeeded]` data object. The logic OR-operator above this data object implies that `PatientFile[medicationNeeded]` can be the result of three fragments: the *Diagnose*, the *Medication*, or the *CPR* fragment.

In this example fCML, the AND-operator was not applied. This can be used to represent the need for several data objects to enable a fragment, or different data objects being produced by a fragment.

4.2 Syntax and Semantics of a Case Model Landscape

After having discussed fCMLs on an example, we will revisit the elements for building such models in this section. In the end, we defined 12 elements for fCMLs. Table 3 provides an overview of these elements and their respective symbols. As previously discussed in Sect. 3.2, we decided to reuse notational elements from BPMN (Business Process Model and Notation) and CMMN (Case Management Model and Notation) – both standards of the Object Management Group having a high recognition factor by business people working with process models. We mostly reuse the notational elements of BPMN and CMMN in such a way that they still have their original meaning.

Formally, a fCML is defined as follows:

Definition 3 An fCML is defined by tuple $cml = (name_{cml}, N, \eta, \tau, R, cs, ce)$ where:

- $name_{cml} : cml \rightarrow String$ is a function assigning each case model landscape a label.
- $N = N_F \cup N_D \cup N_C \cup N_O$ is a non-empty, disjoint set of fragment nodes N_F , data object nodes N_D , event nodes N_C , and logical operator nodes N_O .
 - A data object node $n_D \in N_D$ is defined by the tuple $n_D = (class, state, goal)$ where *class* is the data type, *state* is the current state of the data object, and *goal* $\in \{true, false\}$ indicating whether a data object is part of the goal state.
 - A fragment node $n_F \in N_F$ is defined by the tuple $n_F = (name_F, optional, prerequisite)$ where $name_F : n_F \rightarrow String$ is a function assigning each fragment node a label, $optional \in \{true, false\}$ indicating whether a fragment is optional or non-optional, and

- $prerequisite \in \{blank, conditional, message\}$ assigning a fragment a prerequisite, which defines the enablement conditions of a fragment.
- $\eta : N_C \rightarrow \{start, end, message\}$ is a function that assigns each event node a type.
- $\tau : N_O \rightarrow \{AND, OR, XOR\}$ is a function that assigns the logical operators a type.
- $R \subset (N) \times (N)$ is the causal relation between the nodes.
- $cs \in N_C, \eta(cs) = start$ represents the initial start of the landscape, the case start.
- $ce \in N_C, \eta(ce) = end$ represents the possible termination of a case model because its goal state was reached, the case end.

An fCML should provide a high-level overview of the allowed behavior by the KiP and the possible relationships between its fragments. This will be supported by a graph-based representation showing the abstracted behavior of a knowledge-intensive process from start to end. An fCML for a given fCM case model contains a set of nodes (fragment, data, and event nodes as well as logical operators) and connectors between them. In contrast to an fCM model, the case start and the case end, as well as the prerequisites for fragments are explicitly represented in an fCML.

The semantics of the elements and their relations will be textually described in the following. We also provide the rules for building an fCML based on an fCM case model.

- **fCM landscape.** An fCML encompasses a landscape for a specific fCM case model with the name of the KiP. It consists of a set of nodes, causal relations between these nodes (which will be empty at the beginning), a case start reflecting the initialization of the model, and a case end representing the termination of the model. The fCM landscape is visualized as a rectangular container that is labeled with the name of the corresponding fCM model and includes all its elements.
 - For a given fCM case model, one fCM landscape with the name of the KiP is generated for the respective fCML:

$$cm \Rightarrow cml = (cm.name_{cm}, N, \eta, \tau, R, cs, ce), \text{ whereby } N = \{cs, ce\}$$
 and $R = \emptyset$.
- **Fragment.** An fCML includes a set of fragment nodes. A fragment node is the abstracted representation of the original fragment and is depicted as a round-edged rectangle that shows the name of the fragment and no further details. It has an attribute indicating whether it is optional or not for the case model execution. Optional fragments are visualized with dotted boundaries in comparison to non-optional fragments. The enablement conditions of a fragment are represented as a prerequisite, and further specified based on its different trigger-associations to the other elements. The notation for a prerequisite is a blank diamond located in the boundary of a fragment symbol.

- For each fragment in a given fCM case model, a fragment node with the name of the fragment and a prerequisite from the same type as the start of the fragment is generated in the fCML:

$$\forall f \in cm.F \Rightarrow \{n_F\} \cup N_F \text{ with } (f.name_{fra}, optional, f.s).$$

Whether a fragment is optional is defined as follows:

- *optional* = *false* if the start of the respective fragment can be observed in each possible execution trace of a case model, which leads to the goal state. Please note that a fragment can be in different states (i.e. enabled, disabled, active, finished, and aborted) during its execution as described by Hewelt and Weske in [20]. It is active as soon as it is started by a knowledge worker.
- *optional* = *true* if its start cannot be observed in each execution trace of a case model, which leads to the goal state.

If a case fragment has a blank start event, the case start of the fCML is linked via a causal relation to the corresponding fragment node (visually a connection with the prerequisite of the fragment node):

$$\forall n_F \in N_F, n_F.prerequisite = blank \Rightarrow \{(cs, n_F)\} \cup R.$$

If more than one fragment with a blank start event exists, then a logical operator, an OR, will be added between the case start and the fragment nodes, such that:

$$|(cs, n_i)| > 1, i = \{n_{F1}, \dots, n_{Fn}\} \Rightarrow (\{n_{OR}\} \cup N_{OR}, \tau(n_{OR}) = OR \wedge \{(cs, n_{OR})\} \cup R) \wedge (\forall n_i \Rightarrow \{(cs, n_i)\} \setminus R \wedge \{(n_{OR}, n_i)\} \cup R).$$

It shows that several fragments can be executed right after the start but do not need to. The knowledge worker can decide which of the fragments are executed.

- **External trigger.** Fragments can be enabled by external events, such that they are ready to be executed once the event has occurred. Relevant external events for the start of fragments are represented in the fCM landscape as an event node, depicted as BPMN message start event (unfilled circle with a letter shape inside).
- Each fragment node with an external trigger as prerequisite, an event node of type message is generated, and the event node is connected with the fragment node:

$$\forall n_F \in N_F, n_F.prerequisite = message \Rightarrow \{n_C\} \cup N_C, \eta(n_C) = message \wedge \{(n_C, n_F)\} \cup R.$$

- **Data object.** A data object node in an fCML shows a data condition (i.e., data of a specific type in a specific state) of the fCM case model that is of relevance for the fCML. Namely, if data condition is part of the conditions enabling a fragment (i.e., referenced in the conditional start event of the respective fragment), or is part of the goal state of the fCM case model. If

a data condition is read during the execution of a fragment, which is produced by another fragment (e.g., `PatientFile[diagnosed]` in Fig. 2), it is also represented as a data object node in the fCML because it shows an exchange between fragments. Intra-fragment data is not shown because we focus on the landscape on enablement and interrelations of the fragments. Regarding notation, a data object node is represented by a document-like symbol labeled with the name of the data type followed by its state between square brackets. The symbol is depicted in bold when referred to as the goal state (i.e., in the causal relation with the case end).

- Fragments of the given fCM case model can have a prerequisite consisting of data conditions, which need to be fulfilled for the enablement of a fragment. For each data condition being part of a prerequisite, a data object node is generated in the fCML, which is in a causal relation with the respective fragment node. Thereby, data conditions being part of the same $cond_i \in COND$ of a fragment have to be both available, such that an AND operator is generated between them. Data conditions of different conditions are connected via a OR operator:

$$\begin{aligned} & \forall n_F \in N_F, n_F.prerequisite = conditional \rightarrow (\forall cond \in \\ & \sigma(f.s), f.name_{fra} = n_F.name_F \rightarrow (\forall dc \in cond \rightarrow \{n_D = \\ & (dc.c, dc.[q_c], false)\} \cup N_D \wedge (n_D, n_F) \cup R) \wedge (\exists |(n_i, n_F)| > 1, i = \\ & \{D1, \dots, Dn\} \rightarrow (\{n_{AND}\} \cup N_O, \tau(n_{AND}) = AND \wedge \{(n_{AND}, n_F)\} \cup \\ & R) \wedge (\forall n_i \Rightarrow \{(n_i, n_F)\} \setminus R \wedge \{(n_i, n_{AND})\} \cup R)) \wedge (\exists |(n_j, n_F)| > \\ & 1, j = \{D1, \dots, Dn, AND\} \rightarrow (\{n_{OR}\} \cup N_O, \tau(n_{OR}) = \\ & OR \wedge \{(n_{OR}, n_F)\} \cup R) \wedge (\forall n_j \Rightarrow \{(n_j, n_F)\} \setminus R \wedge \{(n_j, n_{OR})\} \cup R))) \end{aligned}$$

- Analogously to the above, for each data condition defining the goal state of the given fCM case model, a data object is generated in the fCML, which is in a causal relation to the case end:

$$\begin{aligned} & \forall cond \in cm.G \rightarrow (\forall dc \in cond \rightarrow \{n_D = \\ & (dc.c, dc.[q_c], true)\} \cup N_D \wedge (n_D, ce) \cup R) \wedge (\exists |(n_i, ce)| > 1, i = \\ & \{D1, \dots, Dn\} \rightarrow (\{n_{ANDgoal}\} \cup N_O, \tau(n_{ANDgoal}) = \\ & AND \wedge \{(n_{ANDgoal}, ce)\} \cup R) \wedge (\forall n_i \Rightarrow \\ & \{(n_i, ce)\} \setminus R \wedge \{(n_i, n_{ANDgoal})\} \cup R)) \wedge (\exists |(n_j, ce)| > 1, j = \\ & \{D1, \dots, Dn, ANDgoal\} \rightarrow (\{n_{ORgoal}\} \cup N_O, \tau(n_{ORgoal}) = \\ & OR \wedge \{(n_{ORgoal}, ce)\} \cup R) \wedge (\forall n_j \Rightarrow \{(n_j, ce)\} \setminus R \wedge \{(n_j, n_{ORgoal})\} \cup R))). \end{aligned}$$

- For each data node added, the producing fragment node is identified and a causal relation added. If one data condition can be produced from different fragments, then an OR operator is added between the fragments and its representing data object node:

$$\begin{aligned} & \forall n_D \in N_D \rightarrow (\forall f \in cm.F, n_d.dc \in WRITE(f_1) \rightarrow \{(n_F, n_D)\} \cup \\ & R, n_F.name_F = f_1.name_{fra} \wedge (\exists |(n_i, n_D)| > 1, i = \{F1, \dots, Fn\} \rightarrow \\ & (\{n_{ORprod}\} \cup N_O, \tau(n_{ORprod}) = OR \wedge \{(n_{ORprod}, n_D)\} \cup R) \wedge (\forall n_i \Rightarrow \\ & \{(n_i, n_D)\} \setminus R \wedge \{(n_i, n_{ORprod})\} \cup R))) \end{aligned}$$

- For each data condition read by an activity in a fragment, which can also be written by an activity of another fragment of the given fCM case model, a data object node is generated in the fCML. This data

is in a causal relation with the two respective fragment nodes in the fCML:

$$\begin{aligned} \forall dc \in WRITE(f_1) \wedge dc \in READ(f_2), (f_1, f_2) \in cm.F \wedge f_1 \neq f_2 \Rightarrow \\ n_D = (dc.c, dc.[q_c], false) \cup N_D \wedge \{(n_F, n_D)\} \cup R, n_F.name_F = \\ f1.name_{fra} \wedge \{(n_D, n_F)\} \cup R, n_F.name_F = f2.name_{fra}. \end{aligned}$$

- If the same fragment can produce several data conditions having the same data type but in different states, then an XOR operator is added between the fragment node and the alternative data object nodes, which can be produced:

$$\begin{aligned} \exists (|(n_F, n_i)| > 1, i = \{D1, \dots, Dn\} \wedge n_{i1}, n_{i2} \in i, n_{i1}.class = \\ n_{i2}.class) \rightarrow \{(n_{Oprod}\} \cup N_O, \tau(n_{XOprod}) = XOR \wedge \{(n_F, n_{XOprod})\} \cup \\ R) \wedge (\forall n_i \Rightarrow \{(n_F, n_i)\} \setminus R \wedge \{(n_{XOprod}, n_i)\} \cup R) \end{aligned}$$

5 Evaluation

The proposed language for modeling overviews of fragment-based Case Management (fCM) case models – i.e. fCM landscapes (fCMLs) – was evaluated in a laboratory experiment. The goal of the experiment was to assess model interpretation for the aspects of fCM case models that can be represented in an fCML. Consequently, we compared model interpretation of these two types of models to verify whether the use of an fCML would ease model interpretation with respect to an fCM case model. In the experiment, the participants were asked to answer questions on knowledge-intensive processes (KiPs) represented using the different notations. The correctness of the answers, as well as the time needed, was measured to assess *interpretation effectiveness*, *effort*, and *efficiency* as proposed by [5,30]. Open-ended questions were asked to the participants after using each language. The experimental design is described in Sect. 5.1, demographics are shown in Sect. 5.2, quantitative and qualitative analyses are presented in Sect. 5.3 and 5.4, respectively, and discussion is provided in Sect. 5.5.

5.1 Experimental Design

A quasi-experimental design was created for conducting a quantitative as well as a qualitative evaluation of the proposed extension of the fCM-language. The independent variable of the experiment is the modeling language: the proposed extension vs. the fCM-language.

The experiment was a two-treatment factorial crossover design where the experimental object was a two-level blocking variable. In the experiment, each subject had to read an fCM case model for a given KiP (control treatment or C) followed/preceded by reading an fCML of another KiP (experimental treatment or E). The different KiPs used were *traumatology emergency* [35] (experimental object H) and *organization of a business trip* [21] (experimental object B), and their control and treatment model variants, available in the

Appendix of the paper were designed to be informationally equivalent² and were available during the whole experiment as recommended by Parson and Cole [41]. Altogether, this resulted in the following four sequences: EH/CB (sequence A), CB/EH (sequence B), EB/CH (sequence C), and CH/EB (sequence D), as shown in Table 4. For example, in sequence A subjects were exposed to the experimental treatment for the traumatology emergency process in the first period and, to the control treatment for the business trip organization process during the second period.

The experimental subjects were engineering senior undergraduate students with process modeling background from the Chilean university *Pontificia Universidad Católica de Chile*. Each subject was assigned to one of four different groups: each group corresponded to one of the four previously described sequences (see Table 4). Subjects were assigned to the groups according to the initial letter of their last names. This allowed to form four groups of similar size, as shown in Table 4.

Table 4 Crossover design, participants, and valid observations.

Sequence	Period 1	Period 2	No. of participants	No. valid observations
A	EH	CB	9	9
B	CB	EH	13	13
C	EB	CH	12	11
D	CH	EB	13	11

The experiment was conducted using a *Google Forms* questionnaire³. The models and materials of the experiment were designed in Spanish, the native language of the participants and the researchers conducting the evaluation. However, Table 5 provides a translation of the questions of the survey to the English language. The questions consist of a set of demographic questions (items 1-4 in Table 5), a set of true or false statements regarding case model interpretation (items 5-22 in Table 5), two open-ended questions (items 23 and 24 in Table 5), and one open-ended question to assess difficulties regarding the questions (item 25 in Table 5). The statements regarding model interpretation address data and control-flow aspects that are common to an fCM case model and an fCML.

Each respondent was first asked the demographic questions, and were then provided a basic self-training material on CM and the fCM approach by the researchers. Before reading each model, the respective language was explained to the subjects and they were then asked to report the current timestamp. Afterward, subjects were asked to answer the set of questions regarding a model of one of the KiPs, after which they were asked to report once again the current timestamp. The total time used for answering the model interpretation questions was calculated as the difference between the two reported timestamps. Finally, the open-ended questions were asked regarding perceptions in

² W.r.t. aspects measured in the quantitative analysis (see Sect. 5.3).

³ Forms and raw data available at <https://tinyurl.com/u7ogtyb>

using the given language and problems regarding the questionnaire. Then, this was repeated for the other KiP using the other language. The experiment was carried out in a dedicated room during a single two-hour session led by two of the paper authors. At the beginning of the session, they provided each subject with printed versions of the models and an informed consent form. Then, the researchers described the experiment and its instructions, gave access to the questionnaire of the assigned treatment to the participants, and monitored the execution of the experiment. After finishing their participation, each subject would leave the room after handing out – to one of the researchers – their signed informed consent form.

Table 5 Questions of the survey.

Id	Question
1	Besides the <i>Process Modeling</i> course in which you are currently partaking, have you participated in other theoretical or practical process modeling training?
2	Are you familiar with Case Management?
3	How many times have you observed/participated in a trip?
4	How many times have you observed/participated in the provision of medical emergency aid?
5	The first fragment to be executed is [fragment label name].
6	The event [external event label] enables the execution of the fragment [fragment label].
7	The model has a total of [number] fragments.
8	The last fragment of the process is [fragment label].
9	The condition [data condition] enables the execution of the fragment [fragment label].
10	A [data object] is generated by the fragment [fragment label].
11	Having a [data object] is sufficient for enabling the execution of the fragment [fragment label].
12	As soon as the fragment [fragment label] ends, a [data object] is generated.
13	The execution of the fragment [fragment label] changes [data object] into [data object].
14	Having a [data object] is sufficient for enabling the termination of the process.
15	The model has [number] fragment(s) that are executed in every process instantiation.
16	The fragment(s) [fragment(s) label(s)] are not executed in every process instantiation.
17	The fragment [fragment label] can be executed after the fragment [fragment label] has ended.
18	If the fragment [fragment label] has started, it implies that the fragment [fragment label] must have previously ended.
19	Until the fragment(s) [fragment(s) label(s)] have not been executed, the fragment [fragment label] cannot be executed.
20	In some instances of the process, fragment [fragment label] can be executed before fragment [fragment label].
21	The fragment [fragment label] can be executed multiple times within the same process instance.
22	The fragment [fragment label] can be executed more than once during the same process instance.
23	Describe the positive aspects of the language, if any.
24	Describe the negative aspects of the language, if any.
25	Describe difficulties you had with the questions, if any.

The previously described design is based on a pilot study reported in [15]. The main difference between the pilot study and the experiment reported here is that the former was conducted online while the latter was conducted in a controlled environment. The studies also differ regarding subjects and language: while the pilot was run in English with students from the *Hasso Plattner Institute, University of Potsdam, Germany*, the experiment reported here was run in Spanish with students from *Pontificia Universidad Católica de Chile, Chile*. From the experience gained due to the pilot, the following im-

provements were included in the experiment: increased sample size, improved wording of questions, randomization of questions for each form, addition of open-ended questions, and adaptation of models to even complexity between control and experimental treatments⁴.

5.2 Demographics

The subjects of the experiment share a common background due to their enrollment in a *Process Modeling* course when partaking in the experiment. The sample size was 47. However, issues with rebooting of the questionnaire reduced the number of valid observations to 44, as shown in Table 4. The subsequent analysis is concerned with these valid observations, of which demographics are presented in Fig. 3 and Fig. 4. As shown in Fig. 3, nearly half of the subjects claim to have experience regarding process modeling other than the one gained in the course. However, most of the subjects declare no familiarity with case management. As shown in Fig. 4, nearly all subjects declare some experience observing or participating in the organization of a trip, and nearly half in an emergency room as patients.

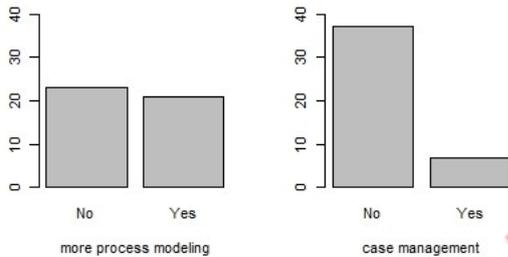


Fig. 3 Process modeling experience.

5.3 Quantitative Analysis

Following the guidelines by Burton-Jones et al. [5], the experiment dependent variables are *interpretation effectiveness* - i.e., how faithfully the interpretation of the model represents the semantics of the model -, *interpretation effort* - i.e., resources needed to interpret the model -, and *interpretation efficiency* - quotient of them both. Interpretation effectiveness was measured as the total score of the set of true-or-false questions (1 point per correct answer), interpretation effort was measured as the total time (in minutes) used by a subject to

⁴ As a complexity measure, we used the number of nodes as proposed by Mendling et al. [34]. The nodes for fCM case models include events, activities, data objects, and gateways. The nodes for fCMLs include all elements in Table 3 except connectors and the fCM landscape.

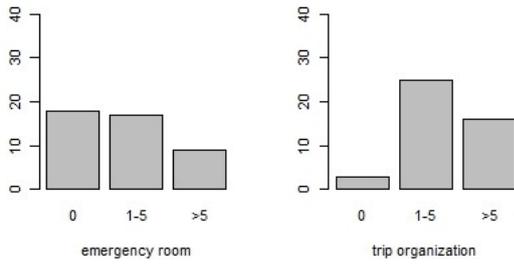


Fig. 4 Process experience.

complete the task, and interpretation efficiency was calculated as the quotient of the previous measurements.

Fig. 5 summarizes the data gathered in the experiment. Overall, data in Fig. 5 for the interpretation effort and efficiency is shifted towards a better performance for our proposal. Regarding the interpretation effectiveness, its average is marginally lower for the extension (15.1/18 points) than for the fCM-language (15.6/18 points). Again on average, better results can be observed for the interpretation effort when using the proposal (10.7 min) in comparison to the fCM-language (13.5 min). On average, regarding interpretation efficiency, the proposal (1.5 points/min) slightly outperforms the fCM-language (1.3 points/min).

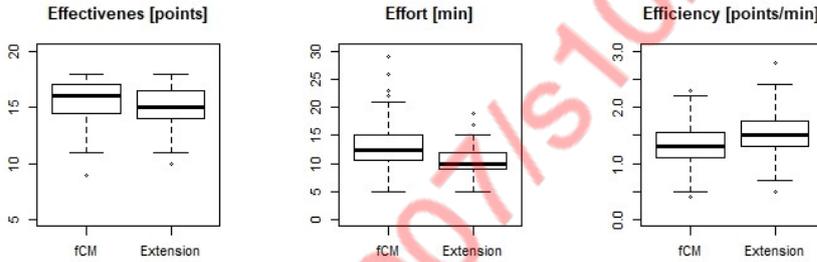


Fig. 5 Descriptive statistics of the experimental dependent variables.

Data was analyzed using a Linear Mixed-Effects (LME) model for taking into consideration data dependency due to repeated measures, as well as random effects due to having repeated measures over subjects [49]. We analyzed the effect of fixed and random factors in each of the dependent variables, namely effectiveness, effort, and efficiency of model interpretation. Fixed factors included treatment ($Treat$), sequence (Seq), period ($Period$), familiarity with a business trip (Fam_B), and familiarity with a traumatology emergency (Fam_H), while the single random factor was subject (s). The model can be described by:

$$Y_{ijk} = \beta_0 + \beta_1 X_{Treat} + \beta_2 X_{Seq} + \beta_3 X_{Period} + \beta_4 X_{Fam_A} + \beta_5 X_{Fam_B} + s_j + \varepsilon_{ijk},$$

where Y_{ijk} is the observation for the i^{th} sequence ($i = A, B, C, D$), j^{th} subject ($j = 1, 2, \dots, n_i$), and k^{th} period ($k = 1, 2$). s_j is a random subject effect term and is assumed to be normal iid, and ε_{ijk} is the error term and is assumed to be normal iid. The assumption for using an LME model is that the residuals for the dependent variable are normally distributed, which was verified via the Shapiro-Wilk test. Residuals conformed to a normal distribution for both effectiveness ($p=0.0056$) and effort ($p=0.0000$). The data for efficiency were transformed via a common logarithm [49], which resulted in a normal distribution of its residuals ($p=0.0042$).

Table 6 Summary of p-values for the ANOVA for each dependent variable.

Variable	Treat	Seq	Period	Fam _B	Fam _H
Effectiveness	0.3415	0.6078	0.0710	0.8898	0.1767
Effort	0.0010	0.6254	0.0000	0.5202	0.1395
Efficiency	0.0249	0.6880	0.0003	0.5944	0.0606

Results of the tests of fixed effects are shown in Table 6. As the table shows, the experiment provided no significant evidence for rejecting that the treatment had no effect on effectiveness ($p>0.05$). This would indicate that a case model that uses our proposal allows users to convey information in a way that interpretation effectiveness does not vary from case models built using the fCM-language. On the other hand, results provided significant evidence for rejecting that the treatment had no effect on effort ($p<0.05$) and efficiency ($p<0.05$). This means that reading case models that use the proposed extension reduces interpretation effort and, altogether, improves interpretation efficiency. Regarding the remaining fixed factors, the tests reported in Table 6 show:

- Sequence had no significant effect on effectiveness ($p>0.05$), effort ($p>0.05$), nor efficiency ($p>0.05$).
- Period was significantly affecting effort ($p<0.05$) and efficiency ($p<0.05$) but not effectiveness ($p>0.05$).
- Familiarity (either with business trip or traumatology emergency) had no significant effect over any analyzed aspect of model interpretation ($p>0.05$).

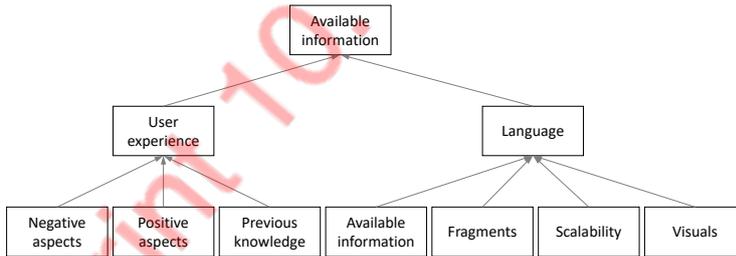


Fig. 6 Common portion of the affinity diagrams.

5.4 Qualitative Analysis

We used the affinity diagramming technique [28] for analyzing open-ended questions of the questionnaire (items 23-25 of Table 5). Affinity diagramming is used for organizing large amounts of unstructured, far-ranging, and seemingly dissimilar qualitative data [18]. The goal of the technique is to build a hierarchy in which each level corresponds to a (sub)category of relevant information. The resulting hierarchy can be graphically depicted as an affinity diagram, where nodes represent categories identified from the qualitative data, and the arcs represent hierarchical relations between the categories. Our aim for using affinity diagramming was to build an affinity diagram for structuring qualitative data gathered for each treatment of the experiment, i.e. data from open-ended questions of the survey. Fig. 6 shows the firsts levels of the affinity diagram, which was common for both treatments. In the following level, the affinity diagrams of each treatment differ, and these differences are explained in detail later on.

In the present work, data were analyzed in Spanish: the native language of both participants and the researchers conducting the analysis. As preparation, data were anonymized by assigning an identifier to each subject (P1, P2, ..., P44). Then, data were divided into two: those questions about the control treatment, and those about the experimental treatment. Four of the paper authors took part in the subsequent analysis which included creating handwritten affinity notes, clustering them and building hierarchies (see Fig. 7), and documenting the results. Two researchers focused on control treatment data, and two on the experimental treatment data. Each group was composed of one female and one male researcher, one researcher more experienced in affinity diagramming, and one more experienced in fCM. This stage of the analysis allowed building one affinity diagram for each treatment, and finalized with each group presenting their affinity diagram to the other. The aforementioned was achieved during a five-hour single session of work held in two dedicated rooms. Afterward, one researcher consolidated the work to make common categories. As a result, two main categories were identified: *user experience* and *language*. In the following, we describe these categories, and up to two levels of their sub-categories. Quotes from participants (translated to English) are included for illustrating our findings. The quotes include the participant identifier followed by a C for control treatment or an E for experimental treatment. Since different people may have different perceptions of the same experience, it is sometimes the case that something that is considered positive by one participant, is considered negative by another participant. This type of data is to be expected in an affinity diagram and it reflects the variety of human experiences.



Fig. 7 Affinity diagramming session.

User experience. Participants identified a number of negative and positive issues regarding their experience in using each language. Also, they addressed the role of previous knowledge, namely the impact of similarities with BPMN.

- **Negative aspects.** Negative aspects of the control treatment were *difficult to understand at first* and *difficult to get an overview*. Some participants claimed problems with the process goal, e.g. “[having separate fragment models] might lead to losing focus on the goal of the process” (P16C). Yet, more often, participants complained about not understanding the process flow, e.g. “it is harder to grasp the process flow” (P5C). Linked to these issues, a number of participants suggested different solutions, e.g. providing a spatial order of the fragments (P12C), using codes for naming the fragments (P33C), providing a complementary flow diagram (P20C). On the other hand, negative aspects for the experimental treatment include *difficult to read* and *difficult to follow*, e.g. “it is hard to follow the flow, it is not intuitive to read, it could be more ordered” (P2E). Difficulties in reading the model involved the following: line-crossover, concurrence modeled with logic operators, spatial distribution of the model, and process complexity.
- **Positive aspects.** Positive aspects of the control treatment include *visual simplicity*, *easy for finding information*, and *efficiency*. Regarding visual simplicity, frequent adjectives for describing the models were “ordered” (e.g. P12C) and “straightforward” (e.g. P2C). Participants pointed out that having separate models of the fragments facilitated finding some information about the modeled processes, particularly details about individual fragments, e.g. “it is clear which are the requisites for executing other [fragments]” (P40C). Also, modularization was perceived as an efficient way for representing a KiP, e.g. “[it] allows to represent complex models in a simple way” (P28C). It was also mentioned that the language allowed for an

“effective visual communication” (P8C) by not representing the KiP in an imperative model and rather showing the flexible nature of these processes. On the other hand, positive aspects of the experimental treatment include *easy to understand* and *easy to follow*, e.g. “it is easy to understand and follow” (P37E). The idea of seeing the process as a whole and being able to identify a sequence seems to impact understandability for the better, e.g. “it is good that fragments are related, since you can see the entire process in a straightforward way” (P12E) and “it is simple to follow the sequence” (P32E).

- **Previous knowledge.** For both treatments, the *reuse of BPMN notation elements* was acknowledged “[it is] similar to BPMN” (P31E); sometimes with a positive connotation, e.g. “it recycles BPMN elements that are familiar” (P38C).

Language. In terms of the language itself, participants referred to aspects related to available information, fragments, scalability, and visuals.

- **Available information.** A small number of participants mentioned issues related to the information included in the models, e.g. “too much information in the fragment [models]” (P13C) and “although some contextual details allow to infer [some information], I believe [the model] does not make them explicit” (P15E).
- **Fragments.** For both treatments, some participants claimed that the optionality of fragments was straightforward and some claimed it was problematic. For the experimental treatment, however, many participants regarded optionality as being straightforward, e.g. “it is easy to see what is optional and what is not” (P30E). A shared issue among treatments was the lack of clarity regarding fragment repetition, e.g. “it is not clear to me, which fragments can be executed more than once and which cannot” (P6C). For both treatments, some participants reported being able to understand relationships between fragments easily, e.g. “It is clear and evident which are the pre-requisites for other activities, which makes it easy to understand how fragments are related” (P40C); whereas others reported the opposite, e.g. “I believe that the big [spatial] separation between fragments makes it confusing for following the process flow” (P43E).
- **Scalability.** Participants were concerned about the issue of scalability in both treatments, especially in the experimental treatment. They envisioned that an increased complexity in the KiP might lead to more complex models, e.g. “I have the feeling that, in a larger model, the number of pre-requisites will increase the number of arcs, and thus increase graphic complexity of the model” (P28E).
- **Visuals.** This sub-category was only identified for the experimental treatment. Most comments in this regard address the concrete syntax. Negative comments include questioning the need for the pre-requisite symbol and

stating limitations of arrows (e.g. P6E), and complaints about the size of logic operators (e.g. P26E). On the other hand, positive comments are concerned with appreciating the symbols of data objects and logic operators (e.g. P2E), and the differentiation of optional fragments (e.g. P9E). It was also mentioned that “[the experimental treatment] has less distracting factors” (P28E).

5.5 Discussion

A number of participants of the experiment shared the opinion that fCM case models support a representation of a KiP that is true to their flexible essence. This backs up the fCM approach as a tool for supporting the modeling of KiP. However, also many participants suggested having another artifact (e.g. flow model or index) or mechanism (e.g. fragment coding) that would somehow be a complement of the fCM case model and/or provide an overview of the KiP. This evidence supports the idea that the fCM approach also has room for improvement, as experienced in [21]. In this regard, an fCML comes to fill a need grounded on empirical evidence.

Quantitative results of our experiment indicate that an fCML maintains the interpretation effectiveness and improves the interpretation efficiency with respect to its equivalent fCM case model. Due to its design, an fCML is a more abstract model and, thus, some information of its equivalent fCM case model is purposeful omitted, e.g. details about fragments. Consequently, the aforementioned findings, are valid only regarding information available in both model kinds. Altogether, quantitative findings support that our proposal improves the interpretation performance of case models within the previously stated limitations. Quantitative results offer some orientation for the reasons behind the obtained quantitative results.

An aspect that might explain the improved effectiveness of fCMLs is the positive perception of some participants regarding their sequence-flow representation of a KiP. Although an imperative model of a KiP carries along some limitations (to be discussed in the following), the subjects claimed to be more familiar with these types of process models. In the same direction, it was also mentioned that a positive feature of fCMLs is that it facilitates seeing the process as a whole. This might be attributed to having all fCM *fragment models* and *goal state* consolidated in a single model.

As highlighted by the participants, the proposed extension of the fCM-language allows depicting some information about the process in a more straightforward way. For example, its notation supports the differentiation of (non)optional fragments by the (non)dotted borderline of the fragment symbol. In contrast, fragment optionality is not directly available in an fCM case model, but rather requires reading all fragment models, raising the cognitive effort. Mechanisms that allow readers to obtain information in a more straightforward manner, might play a key role in reducing interpretation effort and are thus desirable in abstract models.

Regardless of the results of the experiment regarding interpretation performance, participants reported mixed perceptions for ease/difficulty of understanding and following the different model types and the thereof contained fragment relations. These perceptions can be rooted in model quality issues. On one hand, collapsing fragment models (from an fCM case model) into a single symbol (in an fCML), would reduce model complexity because of reducing the number of nodes as discussed by Mendling et al. [34]. But this might not be always the case. In fact, some participants claimed to have difficulties with the simultaneous representation of the alternative flows in fCMLs. Since KiPs allow for many possible execution traces, an imperative model can rapidly increase in complexity as a consequence of the increased number of logic operator nodes. Also, a larger amount of logic operator nodes carries an increase in the number of arcs, which might also lead to undesired line cross-overs. The concern of model scalability emerged for both model kinds, but more frequently regarding fCMLs. This can be attributed to the negative relationship between the degree of a node (i.e. total number of incoming and outgoing arcs) and model understandability as discussed by Mendling et al. [33]. Aspects of the position of the elements and layout, in general, were also mentioned as impacting model readability: these are parameters that have been discussed in the literature of model quality, cf. [25].

5.6 Threats to Validity

We are mindful that our study has some strengths and also some limitations. First, we will discuss validity threats related to experimental subjects. While students are a model of the context, we claim that the subjects of the experiment represent the original context to some extent [23] due to their current enrollment in a *Process Modeling* course. However, and as it has been pointed out in Sect. 5.2, most subjects had no previous background on CM. This constitutes a bias of the experiment, since subjects might have the tendency to prefer procedural representations of processes. An alternative for overcoming these limitations in future work would be providing a more extensive tutorial on CM prior to the experiment.

Second, we discuss validity threats related to experimental design. We decided to use a crossover design because differences between people might lead to large variability when applying a given technique [49]. Crossover designs account for this by using each subject as its own control.

Though we took a number of measures to avoid period-related threats, the data revealed an effect of the period over interpretation effort and efficiency, as shown in Sect. 5.3. We used two experimental objects to prevent the *object learning effect* threat that might have resulted if the same process was used in both treatments. Also, we avoided the threat of *technique learning effect* by asking each subject to apply each treatment only once. We were also able to avoid *copying* and *history* threats due to the experimental setup. Regarding the threat of *fatigue effect*, we decided to tackle it by matching the experiment

session with a *Process Modeling* class of the subjects. However, the lack of a pause between periods might have led subjects to become tired and/or bored, influencing the results to some extent.

As discussed in Sect. 5.3, the sequence had no significant effect on the analyzed aspects of model interpretation, which allowed to discard having an optimal-sequence threat. Similarly, familiarity with the KiP had no significant effect on the analyzed aspects of model interpretation, discarding that prior knowledge of the participants played a role in the results of the experiment. An inherent validity threat of a crossover design is *carryover*, which occurs when a treatment is administered before the effect of a prior treatment has completely receded [49]. However, we infer that neither carryover nor period*treatment interaction influenced the analyzed aspects of model interpretation, since sequence – innocuous as previously discussed – is intrinsically confounded with carryover and the period*treatment interaction in our experiment.

Finally, we would like to discuss the strengths and limitations of the qualitative data analysis, namely the affinity diagramming. An inherent limitation of affinity diagrams lies in not being fully reproducible due to the subjectivity of researchers in grouping affinity notes and defining categories. A benefit of the technique – versus, e.g., the use of spreadsheets – is that researchers are able to analyze data jointly, leading to counterbalance for individual subjectivity. A challenge linked to the aforementioned, however, is the need to build balanced groups of researchers, which is the reason why we worked in the pairs described in Sect. 5.4.

6 Conclusions

This paper provided a new concept for case management by presenting a means for modeling case model landscapes. This contribution is built upon extending the fragment-based Case Management (fCM) language. A case model landscape (CML) gives end users an integrated, comprehensive overview of the high-level activities and the processed data during the execution of a knowledge-intensive process instead of the detailed case models with often scattered information about actions and data in different models. It can be used to get an understanding, but also to analyze case models, redesign, or check compliance requirements. As the landscapes build upon the fCM approach, we focus on fCM landscapes (fCMLs). We tested the interpretation performance of our proposal in an experiment with students. Results implicate that the proposal while maintaining the effectiveness of model interpretation, improves its efficiency. These findings should be considered within the limitations discussed in Sect. 5.6. Also, we are aware that further requirements for defining fCMLs might be identified in future work. However, the aim of this paper was not to conduct an exhaustive search for requirements, but rather to focus on solving the understandability of fCM case models.

The proposed fCM-language extension for CML reuses notational elements of the two modeling standards, BPMN (Business Process Modeling and Nota-

tion) and CMMN (Case Management Model and Notation). While this has the advantage that it might be easier understandable by business people working with process models, it has the risk of some minor misinterpretation which need to be further tested.

Our proposal addresses the fCM approach, however, we hereafter discuss how it might be extended to other approaches. We also discuss how other approaches might enrich our work. The proposal could be also used for CMMN models, whereby stages and their relationship could be shown on an abstract level. CMMN represents data mainly implicitly, our language represents data and data relations explicitly. Furthermore, the approach might be also interesting for PHILharmonicFlows, another relevant case management approach, to represent the relation between the micro processes. An important concept for PHILharmonicFlows are the cardinalities between the generated objects. These are only implicitly given in the proposed landscape by distinguishing between optional and mandatory fragments, and the possibility to enable certain fragments more than once. An explicit representation might be a useful extension.

Overall, we believe that the understandability of case models benefits from the use of complementary models with different levels of abstraction. This allows for distributing the inherent complexity of KiPs in views that can be more adequate for solving specific model-related tasks. For the fCM approach, fCM case models (focusing on the details of fragments) are complemented with the more abstract fCMLs (focusing on the process overview). However, and as discussed earlier, the scalability of these models is still an open challenge. In this regard, we believe that future research might explore a dynamic visualization of fCMLs that allows readers to select portions of the model they want to see. Other aspects that we recommend to consider for improving case model understandability is making use of familiar notation – such as reuse of BPMN concrete syntax – and structures – such as workflow patterns. Additionally, future work could consider additional techniques for assessing the model understandability, such as measures of mental effort, e.g., with the help of eye trackers as suggested by [50].

Acknowledgments

This work is supported by the Beca Postdoctorado Escuela de Ingeniería, Pontificia Universidad Católica de Chile, and CORFO Engineering 2030 14ENI2-26862. The work is also supported by CONICYT FONDECYT projects 11170092, 1200206, and 1181162. We thank Javier Madariaga for his help in reviewing part of the statistical analysis. We would also like to sincerely thank the reviewers for their constructive feedback during the review process.

References

1. van der Aalst, W., Berens, P.: Beyond workflow management: product-driven case handling. In: 2001 International ACM SIGGROUP Conference on Supporting Group Work, pp. 42–51. ACM (2001)
2. van der Aalst, W., Pesic, M., Schonenberg, H.: Declarative workflows: Balancing between flexibility and support. *Computer Science - Research and Development* **23**, 99–113 (2009)
3. van der Aalst, W., Weske, M., Grünbauer, D.: Case handling: A new paradigm for business process support. *Data & Knowledge Engineering* **53**(2), 129–162 (2005)
4. Becker, J., Pfeiffer, D., Räckers, M., Fuchs, P.: Business Process Management in Public Administrations - The PICTURE Approach. In: PACIS 2007, Auckland, New Zealand, July 3-6, pp. 1–14 (2007)
5. Burton-Jones, A., Wand, Y., Weber, R.: Guidelines for empirical evaluations of conceptual modeling grammars. *Journal of the Association for Information Systems* **10**(6), 495–532 (2009)
6. Chiao, C.M., Künzle, V., Reichert, M.: Integrated modeling of process-and data-centric software systems with philharmonicflows. In: 2013 IEEE 1st International Workshop on Communicating Business Process and Software Models Quality, Understandability, and Maintainability (CPSM), pp. 1–10. IEEE (2013)
7. Davis, F.: Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Q.* **13**(3), 319–340 (1989)
8. De Smedt, J., De Weerd, J., Serral, E., Vanthienen, J.: Discovering hidden dependencies in constraint-based declarative process models for improving understandability. *Information Systems* **74**, 40–52 (2018)
9. Di Ciccio, C., Marrella, A., Russo, A.: Knowledge-intensive processes: characteristics, requirements and analysis of contemporary approaches. *Journal on Data Semantics* **4**(1), 29–57 (2015)
10. Dijkman, R., Vanderfeesten, I., Reijers, H.: Business process architectures: Overview, comparison and framework. *Enterp. Inf. Syst.* **10**(2), 129–158 (2016)
11. Eid-Sabbagh, R.H., Dijkman, R., Weske, M.: Business Process Architecture: Use and Correctness. In: BPM 2012, Tallinn, Estonia, September 3–6, pp. 65–81. Springer (2012)
12. Eid-Sabbagh, R.H., Hewelt, M., Meyer, A., Weske, M.: Deriving Business Process Data Architectures from Process Model Collections. In: ICSOC 2013, Berlin, Germany, December 2-5, pp. 533–540. Springer (2013)
13. Gonzalez-Lopez, F., Bustos, G.: Business process architecture design methodologies – a literature review. *Business Process Management Journal* **25**(6), 1317–1334 (2019)
14. Gonzalez-Lopez, F., Bustos, G.: Integration of Business Process Architectures within Enterprise Architecture Approaches: A Literature Review. *Engineering Management Journal* **31**(2), 127–140 (2019). DOI 10.1080/10429247.2018.1522565
15. Gonzalez-Lopez, F., Pufahl, L.: A landscape for case models. In: Enterprise, Business-Process and Information Systems Modeling, pp. 87–102. Springer (2019)
16. Green, S., Ould, M.: The Primacy of Process Architecture. *CAiSE Workshops* (2) pp. 154–159 (2004)
17. Gruhn, V., Wellen, U.: Analysing a Process Landscape by Simulation. *Journal of Systems and Software* **59**(3), 333–342 (2001)
18. Hartson, R., Pyla, P.: *The UX Book: Process and Guidelines for Ensuring a Quality User Experience*. Morgan Kaufmann, Amsterdam (2012)
19. Hewelt, M., Pufahl, L., Mandal, S., Wolff, F., Weske, M.: Toward a methodology for case modeling. *Software and Systems Modeling* pp. 1–27 (2019)
20. Hewelt, M., Weske, M.: A Hybrid Approach for Flexible Case Modeling and Execution. In: M. La Rosa, P. Loos, O. Pastor (eds.) *Business Process Management Forum. BPM 2016.*, vol. 260. Springer (2016)
21. Hewelt, M., Wolff, F., Mandal, S., Pufahl, L., Weske, M.: Towards a Methodology for Case Model Elicitation. In: Enterprise, Business-Process and Information Systems Modeling. *BPMDS 2018, EMMSAD 2018*, vol. 318. Springer (2018)
22. ter Hofstede, A.H., Proper, H.A.: How to formalize it?: Formalization principles for information system development methods. *Information and Software Technology* **40**(10), 519–540 (1998)

23. Höst, M., Regnell, B., Wohlin, C.: Using Students as Subjects—A Comparative Study of Students and Professionals in Lead-Time Impact Assessment. *Empirical Software Engineering* **5**(3), 201–214 (2000). DOI 10.1023/a:1026586415054
24. Hull, R., Damaggio, E., De Masellis, R., Fournier, F., Gupta, M., Heath III, F.T., Hobson, S., Linehan, M., Maradugu, S., Nigam, A., et al.: Business artifacts with guard-stage-milestone lifecycles: managing artifact interactions with conditions and events. In: DEBS 2011, pp. 51–62. ACM (2011)
25. Krogstie, J.: Quality of Business Process Models. In: Proceedings of PoEM’2012. LNBIP vol. 134, pp. 76–90 (2012)
26. Künzle, V., Reichert, M.: PHILharmonicFlows: Towards a framework for object-aware process management. *Journal of Software Maintenance and Evolution: Research and Practice* **23**, 205–244 (2011)
27. Lantow, B.: Adaptive case management—a review of method support. In: POEM 2018, pp. 157–171. Springer (2018)
28. Lucero, A.: Using Affinity Diagrams to Evaluate Interactive Prototypes. In: INTERACT 2015, pp. 231–248. Springer (2015)
29. Lunn, K., Sixsmith, A., Lindsay, A., Vaarama, M.: Traceability in requirements through process modelling , applied to social care applications. *Information and Software Technology* **45**(15), 1045–1052 (2003)
30. Malinova, M.: A Language for Designing Process Maps. Ph.D. thesis, Vienna University of Economics and Business (2016)
31. Malinova, M., Leopold, H., Mendling, J.: An Explorative Study for Process Map Design. In: CAiSE Forum, June 16–20, Thessaloniki, Greece (2014)
32. Marin, M.A., Hauder, M., Matthes, F.: Case Management: An Evaluation of Existing Approaches for Knowledge-Intensive Processes. In: BPM Workshops 2015. Springer (2015)
33. Mendling, J., Reijers, H.A., van der Aalst, W.: Seven process modeling guidelines (7PMG). *Information and Software Technology* **52**(2), 127–136 (2010)
34. Mendling, J., Reijers, H.a., Cardoso, J.: What makes process models understandable? *Business Process Management Journal* pp. 48–63 (2007)
35. Mertens, S., Frederik, Poels, G.: Enhancing Declarative Process Models with DMN Decision Logic. In: Enterprise, Business-Process and Information Systems Modeling. BPMDS 2015, EMMSAD 2015, pp. 151–165. Springer (2015)
36. Moody, D.: The “Physics” of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. *IEEE Transactions on Software Engineering* **35**(6), 756–779 (2009)
37. Nigam, A., Caswell, N.S.: Business artifacts: An approach to operational specification. *IBM Systems Journal* **42**(3), 428–445 (2003)
38. Norman, D.A.: Cognitive engineering. User centered system design pp. 31–61 (1986)
39. OMG: Business Process Model and Notation (BPMN), V. 2.0 (2011)
40. OMG: Case Management Model and Notation (CMMN) V. 1.1 (2016)
41. Parsons, J., Cole, L.: What do the pictures mean? guidelines for experimental evaluation of representation fidelity in diagrammatical conceptual modeling techniques. *Data & Knowledge Engineering* **55**(3), 327–342 (2005)
42. Pestic, M., van der Aalst, W.: A declarative approach for flexible business processes management. In: BPM 2006, pp. 169–180. Springer (2006)
43. Pichler, P., Weber, B., Zugal, S., Pinggera, J., Mendling, J., Reijers, H.A.: Imperative versus declarative process modeling languages: An empirical investigation. In: BPM 2011, pp. 383–394. Springer (2011)
44. Pufahl, L., Ihde, S., Glöckner, M., Franczyk, B., Paulus, B., Weske, M.: Countering congestion: A white-label platform for the last mile parcel delivery. In: BIS 2020. LNBIP vol. 389. Springer (2020)
45. Steinau, S., Andrews, K., Reichert, M.: The relational process structure. In: CAiSE 2018, pp. 53–67. Springer (2018)
46. 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**, 2679–2716 (2019)
47. The Open Group: ArchiMate 3.0.1 Specification (2017)

48. Vaculin, R., Hull, R., Heath, T., Cochran, C., Nigam, A., Sukaviriya, P.: Declarative business artifact centric modeling of decision and knowledge intensive business processes. In: 15th IEEE International Conference on Enterprise Distributed Object Computing (EDOC 2011), pp. 151–160. IEEE (2011)
49. Vegas, S., Apa, C., Juristo, N.: Crossover designs in software engineering experiments: Benefits and perils. IEEE Transactions on Software Engineering **42**(2), 120–135 (2016)
50. Weber, B., Neurauter, M., Pinggera, J., Zugel, S., Furtner, M., Martini, M., Sachse, P.: Measuring Cognitive Load During Process Model Creation. In: Information Systems and Neuroscience. LNISO vol. 10. Springer (2015)
51. Weske, M.: Business process management: Concepts, Languages, Architectures. Springer (2012)
52. Wieringa, R.J.: Design Science Methodology for Information Systems and Software Engineering, 1 edn. Springer-Verlag Berlin Heidelberg (2014). DOI 10.1007/978-3-662-43839-8
53. Zensen, A., Küster, J.: A comparison of flexible BPMN and CMMN in practice. In: EDOC 2018, pp. 105–114. IEEE (2018)

Appendix

In the following, Fig. 8 to 11 show the models used in the experiment.

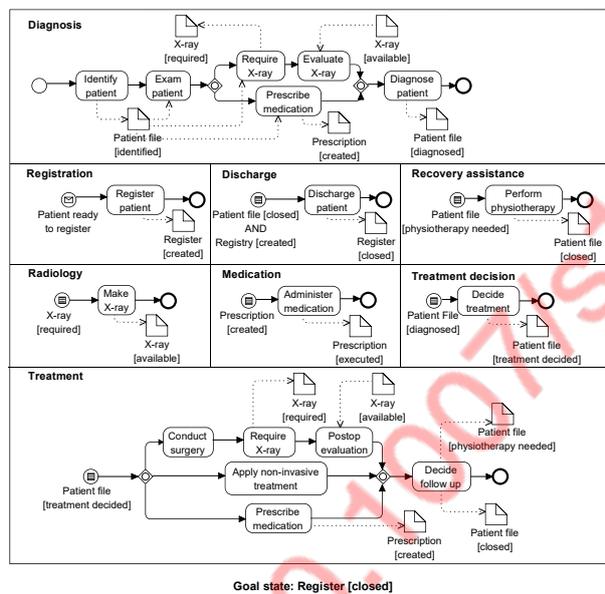


Fig. 8 fCM case model for *traumatology emergency*.

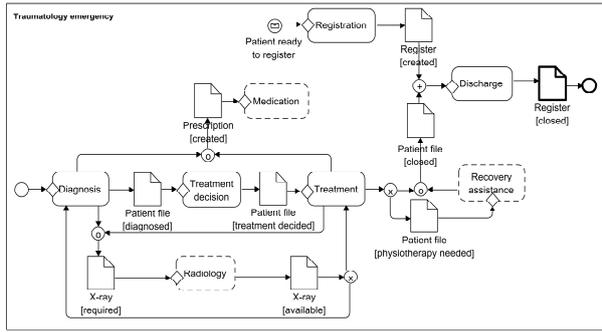
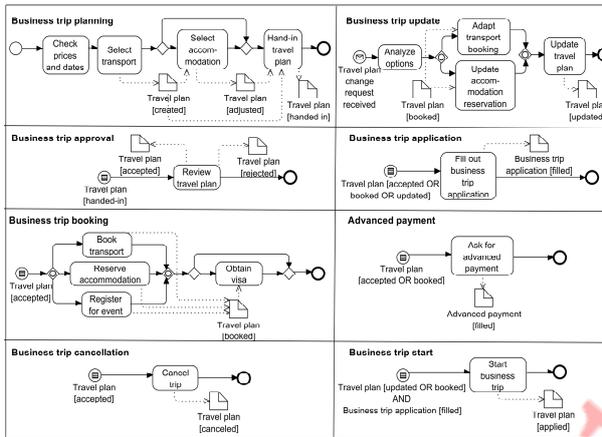


Fig. 9 fCML for *traumatology emergency*.



Goal state: Travel plan [applied OR canceled OR rejected]

Fig. 10 fCM case model for *organization of a business trip*.

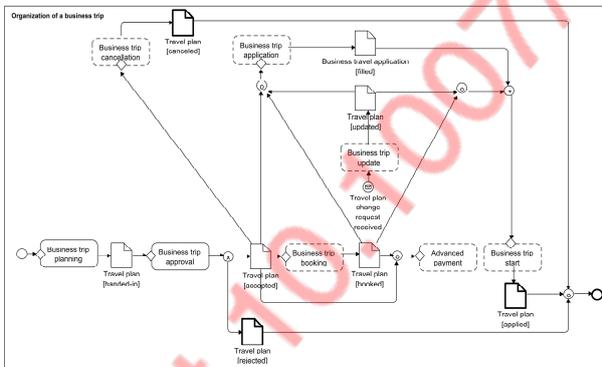


Fig. 11 fCML for *organization of a business trip*.