Data-centric goal modeling for knowledge-intensive processes

Anjo Seidel[1,*], Charlotte Balcke[1] and Mathias Weske[1]

[1]Hasso Plattner Institute, University of Potsdam, Prof.-Dr.-Helmert Str. 2–3, Potsdam, 14482, Germany

Abstract

Knowledge-intensive processes are both data-centric and goal-oriented. Data is manipulated by processes, and knowledge workers can terminate a process once its goal has been reached. Despite the fact that formally defined goals are crucial for effective process executions, no visual notation for modeling goals in knowledge-intensive processes has been introduced yet. This paper proposes to model goals graphically as UML object diagrams that are consistent with the UML class diagrams describing the data, which is manipulated by knowledge-intensive processes. A prototypical implementation provides a visual modeler for goals in knowledge-intensive processes.

Keywords

Business Process Management, Knowledge-intensive Processes, Data-centric Processes, Goal Modeling, Fragment-based Case Management

1. Introduction

In business process management (BPM), process models are used to depict and support real-world processes [1]. A special class of business processes is knowledge-intensive processes (KiPs), which are especially unpredictable, emergent, and complex [2]. Knowledge workers drive those processes and aim to attain specific goals, which are desirable states of a business process. In order to reach such a goal, knowledge workers plan their actions, rendering decision-making an essential part of their work [3]. Actions for the current situation are chosen in regard to process instance-specific goals [4]. Often it is not trivial to identify actions that align with one's goals and knowledge workers can benefit from recommendations for the next best actions [5]. Furthermore, automatic planning can support knowledge workers but requires defined process instance goals as well [4].

Different data-centric modeling approaches, like BAUML [6], OCBC [7], and fragment-based case management (fCM) aim to model KiPs. They model involved data objects using UML class diagrams as domain models and combine them with behavioral models. Goals need to be concerned with the data that is manipulated during process executions.

Goals for knowledge-intensive processes can be defined as a combination of objectives. Objectives are sub-goals that make quantifiable statements about a case [8]. They can be

ER2023: Companion Proceedings of the 42nd International Conference on Conceptual Modeling: ER Forum, 7th SCME, Project Exhibitions, Posters and Demos, and Doctoral Consortium, November 06-09, 2023, Lisbon, Portugal

*Corresponding author.

anjo.seidel@hpi.de (A. Seidel); charlotte.balcke@student.hpi.de (C. Balcke); mathias.weske@hpi.de (M. Weske)

0000-0002-9652-5340 (A. Seidel); 0000-0002-3346-2442 (M. Weske)

combined into long-term goals. Goals that are within the scope of an fCM model can be formally defined as partial orders of objectives [5]. Objectives are first-order logic statements over single execution states of the process model. Until now, no graphical notation for such goal models was provided, which could ease the accessibility for modelers compared to formal expressions.

In this paper, we propose a graphical visualization of data-centric goals for knowledge-intensive processes and the means for formal verification of goals and process models. This enables knowledge workers to model their desired process outcomes flexibly and receive recommendations accordingly. Based on fCM, we show how (i) data-centric objectives can be modeled with UML object diagrams, which are instances of the process model's domain model. (ii) Goals can be represented as dependency graphs that order objectives. We provide an interpretation of these graphical goals that conforms to the one by Seidel et al. [5], so they can be used for the same formal verification and computation of planning support. A prototypical implementation allows for visually modeling data-centric goals that are compliant with fragment-based case models. The prototype allows for deriving recommendations by filtering suitable from non-suitable actions with regard to a modeled goal.

In the remainder of this paper, we present the related work section 2, before introducing the preliminaries to our approach in section 3. In section 4, the notation and semantics of data-centric goals are elaborated. A prototypical implementation is presented in section 5, before section 6 concludes the paper.

2. Related work

Our contribution is based on different previous streams of work regarding KiPs and KiP models, and goal modeling for BPM.

Di Ciccio et al. provide a definition and a set of characteristics for KiPs and requirements towards modeling them [2]. KiP models need to support data and goal modeling during run-time. Meanwhile, Pyoria et al. define decision-making during process execution as a crucial task of knowledge workers [3].

Declarative and data-centric process modeling approaches have been developed to model KiPs. Di Ciccio et al. [2] and Steinau et al. [9] provide an overview of contemporary approaches. The data-centric approaches BAUML [6], OCBC [7], and fragment-based Case Management (fCM) [10, 11] combine behavioral models with UML class diagrams to model the involved data objects.

Goal modeling is a prominent topic in the context of requirements engineering. Kavakli et al. [12] and Amyot et al. [13] provide an overview of existing methods. Goal modeling was combined with BPM in many prior works. Kueng and Kawalek [14] provide an overview of goal-based business process modeling. Greenwood and Rimassa [15] provide a framework for automatic goal-oriented BPM. Amyot et al. [16] analyze and summarize existing combinations of business process modeling with goal modeling in the URN standard. They summarize the main application fields of aligning process models with goals, process compliance, adaptation and improvement, and goal-oriented process mining. Other works allow developing process models from i* goal models [17] and from KAOS goal models [18]. Yet, these goal modeling

approaches for BPM focus on requirements for the process model instead of single process instances. All those goal models mainly focus on high-level requirements toward the process model itself during the design-time of process models instead of process instance goals.

Fewer works focus on the usage of process goals during run-time. Dalpiaz et al. [19] extend goal models for monitoring their satisfaction during run-time. Ghanavati et al. [20] summarize the works of using goal models for business process compliance. Marella et al. [21] provide a framework to map business process models to planning problems including goals as desired process outcomes. Based on fCM, Haarmann et al. [22] and Seidel et al. [5] provided the means to formally define the objectives and goals for the outcome of process models during run-time as first-order logic statements over possible future execution states. For now, those goals are only defined formally and not visually.

3. Preliminaries

This paper presents an approach to model goals by using UML object diagrams for data-centric process models. We apply our concepts to the fragment-based Case Management (fCM) approach. An introduction to fCM is included by providing a running example. Furthermore, we summarize preliminary works toward the definition and utilization of goals for fCM. Finally, a short introduction to the relevant UML concepts is given.

3.1. Fragment-based Case Management

The key concept of fCM is the combination of imperative control flow with declarative data flow. The process is split into fragments that can be combined repeatedly and concurrently during run-time. Their combination is constrained by the control flow and by data constraints. An fCM model consists of (i) behavioral process fragments, (ii) a domain model, and (iii) object lifecycles.

Fragments. The process fragments have a similar syntax to BPMN [10]. They define a set of activities. For each activity, the required data objects for execution can be defined. Activities describe possible data changes.

Consider the example of renovating a house. The process can be modeled with the fragments displayed in Figure 1. Once the construction site is opened, the work on the house is started in fragment F1. Then, the house is considered to be *started* and the work on the staircase and on the apartments can start as well. According to one's needs, the house can be split into multiple apartments. In fragment F2, new apartments can be created. For each apartment, multiple tasks need to be done. As specified in fragment F3, for an initial or painted apartment, tiles are laid. In fragment F4, an apartment, initial or with tiles, is painted. It can be painted with either yellow or green paint. An apartment can be painted and tiled concurrently. Afterward, the kitchen can be installed in fragment F5 and the apartment is considered *finished*. The finished apartments are a preliminary in fragment F1 to paint the staircase. Afterward, the house can be checked and accepted.

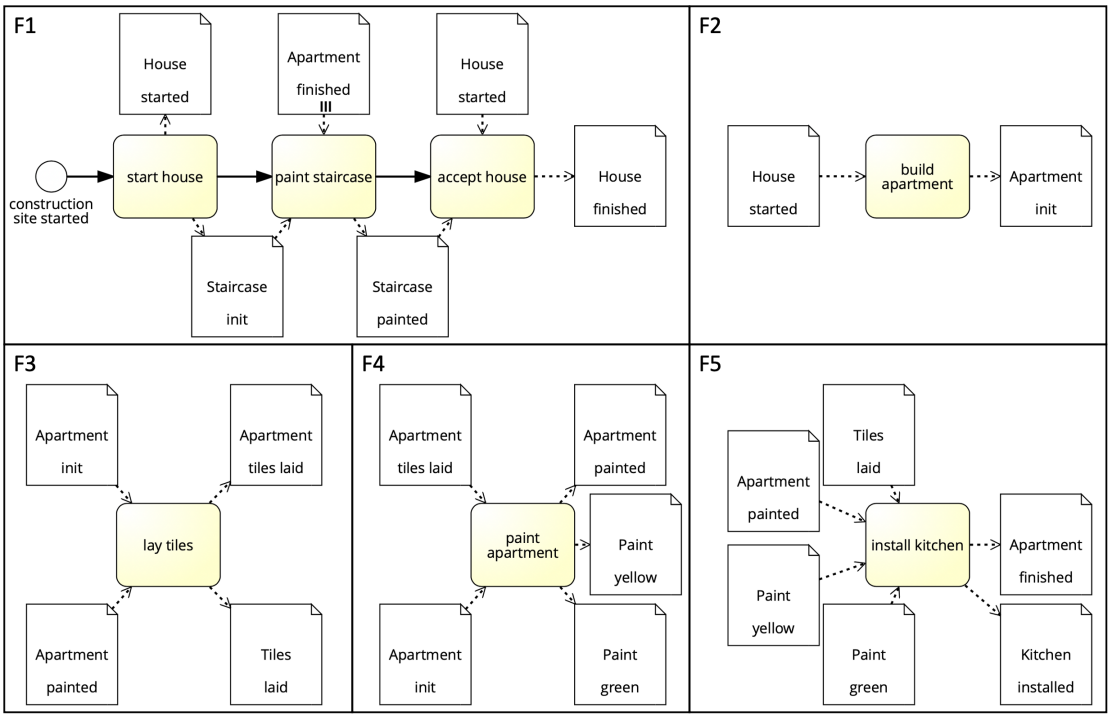

Figure 1: Fragments for the house renovation process.

Domain model. The domain model of the fCM model is modeled with a UML class diagram. It describes the set of data classes C and their associations. Every involved data object in the process is an instance of such a data class. Each execution of a case in fCM evolves around the case object. For the renovation example, the case class is the house as seen in Figure 2. Each house can have a staircase and multiple apartments. For each apartment, a kitchen can be built, paint can be applied once and tiles can be laid multiple times. The domain model yields restrictions for the possible execution of the fragments, e.g., *paint apartment* can only be executed once per apartment.

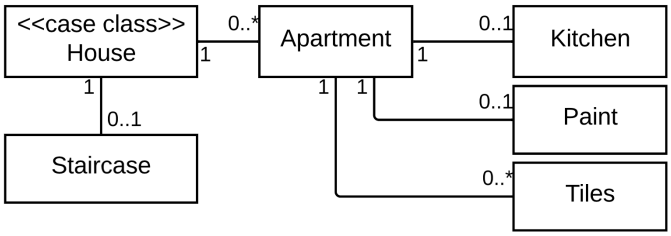

Figure 2: Domain model for the house renovation process.

Object behavior. In fCM, the concrete attribute values of a data object are abstracted as states. During execution, states may change according to their allowed behavior specified by state transition systems for each data class. It defines a set of states S_c for each data class $c \in C$ and possible state changes. The object behavior of the renovation process is displayed in Figure 3. Each house instance can first be *started* before becoming *finished*. A staircase is first *init* and can then be *painted*. An apartment can change its states from *init* to *painted* or *tiles laid*, it can change between the latter two, before being *finished*. Tiles can be *laid*, paint is either *yellow* or *green*, and kitchens can be *installed*.

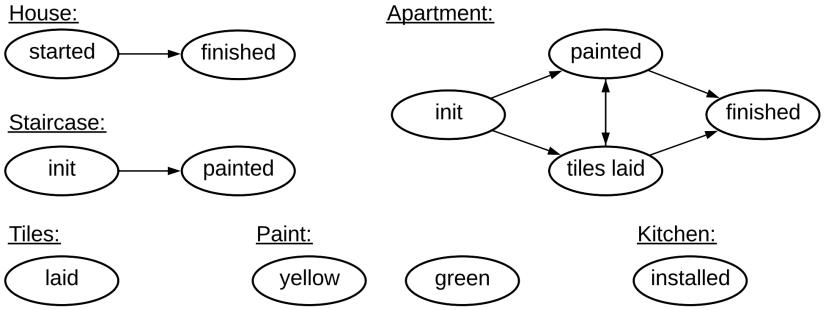

Figure 3: Object behavior for the house renovation process.

Execution semantics. During the execution of a case, the case state defines the currently available data objects, their links, and actions [5]. It consists of the tuple (O, L, A). O is the set of all data objects, such that a data object $o \in O$ is an instance of a data class $c \in C$ in the domain model. It is in a state $s_c \in S_c$ of the classes' state transition system. Furthermore, each data object has a unique identifier $id \in ID$.

For the current state, L holds all the links of the present data objects. A link is an unordered pair of data objects that became associated during the case execution according to the domain model. The possible actions for the current state are present in the set A. An action $a \in A$ is the instance of an activity defined in the fragments.

During renovation, a case may be in a state, where there is a started house with the ID *House 1*. It is linked to two apartments with the IDs *Apartment 1* and *Apartment 2*. Apartment 1 is in init, while Apartment 2 is already painted yellow. Available actions include creating a new apartment, laying tiles for both apartments, and painting Apartment 1 either yellow or green.

3.2. Goal models for fCM

Goal models that are within the scope of an fCM model can be used to derive recommendations that support decisions during run-time [5, 22]. Goals are orders of objectives over time. Objectives and goals can be formally defined as first-order logic statements and partial orders [5].

The set of all possible objectives Ω are all possible first-order logical statements over any case state $s = (O, L, A)$ [5]. An objective $\omega_1 \in \Omega$ may express that in a state s, the *Apartment 2*

should have paint:

$$\omega_1 \equiv \exists (o_1, o_2) \in L \, : \, o_1.class = Apartment \wedge o_1.id = "Apartment2" \wedge o_2.class = Paint$$

Another objective ω_2 states that a painted staircase should exist:

$$\omega_2 \equiv \exists o \in O \, : \, o.class = Staircase \wedge o.state = painted$$

Goals are a partial order of temporally ordered objectives [5]. A partial order is a reflexive, transitive, and antisymmetric relation over a set of elements. In the given definition, goals are defined as a partial order over the set of objectives and via a set of pairs with directly comparable objectives. The goal γ_1 orders the set of objectives $\{\omega_1, \omega_2\}$ partially with the following specification:

$$\gamma_1 = (\{\omega_1, \omega_2\}, \{(\omega_1, \omega_1), (\omega_1, \omega_2), (\omega_2, \omega_2)\}).$$

Those goals can then be utilized to analyze the process model's state space, i.e., all possible behavior, for execution sequences that comply with the defined goals [5].

3.3. UML object diagrams

The unified modeling language (UML) [23] is a well-established standard [24]. In the fields of computer science and business applications, UML is a frequently used modeling tool [25]. Class diagrams are used most prominently, among others to define the domain of data-centric process models like BAUML [6], OCBC [7], and also fCM [11].

While class diagrams can represent the possible data objects for a process, UML object diagrams may represent concrete data instances of such classes. Modelers can define concrete data instances and their relations. In Figure 4, an object diagram represents an instance of the fCM's domain model (cf. Figure 2). A house with concrete attribute values is linked to two apartment instances. They have concrete sizes and numbers of rooms.

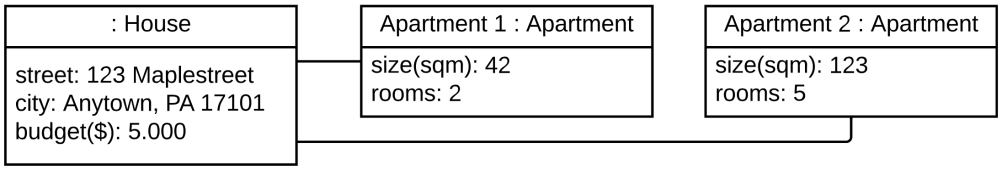

Figure 4: A UML object diagram representing the instances of a house and two apartments.

4. Data-centric goal models

As defined in section 3, we consider goals to be an ordering of objectives over time, while objectives are logical statements about the execution state of a process model. In the following, we propose the notation of UML object diagrams to model objectives consistent with a given data-centric process model. Dependency graphs are used to model the temporal ordering of objectives. For both, we provide formal semantics that maps to the generic definition of goals and objectives as presented in section 3.

4.1. Modeling objectives with UML object diagrams

Notation. Data-centric process models, like fCM, describe all involved data classes and possible relations between them in a domain model as UML class diagrams. UML object diagrams provide the opportunity to model instances of class diagrams at a certain point in time. They express a configuration of instances of the data classes—the set of data objects and their links.

Objectives, on the other hand, are defined as logical statements over the set of data objects and their links for a given point in time. The data objects and links relate to instances of the domain model, i.e., a UML class diagram. The states of data objects relate to the object lifecycle of their respective data class.

Therefore, we utilize UML object diagrams to model data-centric objectives. The UML object diagram acts as an instance of the UML class diagram in the process model. The behavior in an fCM model abstracts from concrete attribute values and operates only on states. Therefore, we also abstract from concrete values for object attributes in the UML object diagram. We utilize the possible states of the data objects according to the object lifecycles as a substitute.

For each data object, a modeler can specify the class of the object. This class relates to one of the data classes $c \in C$ of the domain model. Also, a unique identifier for the object can be defined. Furthermore, the modeler may specify the state or states that the data object is allowed to be in. The state attribute in the object diagram is a list of states S_c for the specified class c. The reasoning for such a list of states is that the object should exist and it should be in one of the modeled states. If the list is empty, the object may be in any state.

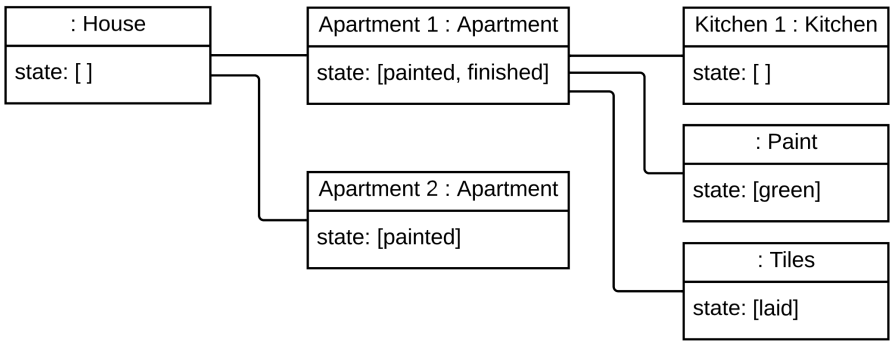

Figure 5: The UML object diagram representing objective ω_3.

In section 3, the objectives ω_1 and ω_2 were already defined as a first-order logic statement. In Figure 5, a new objective ω_3 is defined as a UML object diagram. It requires that there exists an instance of the class *House*, which is linked to two apartments. One apartment shall have the identifier *"Apartment 1"* and the other *"Apartment 2"*. For the first apartment, there shall exist the kitchen *"Kitchen 1"*, it shall be either *painted* or *finished* and have green paint and laid tiles. The second apartment shall be *painted*.

Consistency with the data model. Since objectives define single case states, the UML object diagram representing it must be consistent with the fCM. The object diagram must be a

valid instance of the domain model and its states must comply with the object lifecycles. The following consistency guidelines ensure consistency between the different models.

(i) Every data object must be an instance of a class in the domain model. Yet, not all classes have to be represented.

(ii) The links of objects must be coherent with the class diagram. Objects can only be linked if their classes are associated. For instance, paint and tile instances must not be linked.

(iii) Furthermore, the number of links in the object diagram must be coherent with the multiplicities of the associations in the class diagram. The class *House* is associated with at most four apartments. An objective connecting a house instance with five apartments would violate our domain model.

(iv) Lastly, all object states in the objectives must correspond to states of the object behavior of the corresponding data class. For example, a house instance can only be in the states *started* or *finished*. Like the first guideline, not all states have to be used.

Even if these guidelines are met, it is still possible to model objectives that are impossible to reach because the process model can never reach an according state. To detect this, model checking can be utilized [5].

Formal representation. In order to utilize the proposed UML object diagrams for formal verification of the fCM process model against a goal, we propose a representation of the object diagrams as formal objectives. An objective is a first-order logic formula over an execution state of the process model. This state s holds information about the data objects O and links L.

For each data object node in the object diagram, a statement is created. It determines that there exists an object $o \in O$ in the set of data objects such that the class of o is the same as the object nodes. Considering ω_3 as specified in Figure 5, which requires the existence of a house, is formalized as follows:

$$\exists o \in O \,:\, o.class = House$$

If an identifier is specified in the object node, the identifier of o needs to be the same as the object node's. In ω_3, *Kitchen 1* is required.

$$\exists o \in O \,:\, o.class = Kitchen \wedge o.id = \text{"Kitchen 1"}$$

For a data object node where a state is defined, the data object $o \in O$ needs to have the same state as specified in the node. In ω_3, *Apartment 2* is required to be in the state *painted*.

$$\exists o \in O \,:\, o.class = Apartment \wedge o.id = \text{"Apartment 2"} \wedge o.state = painted$$

For an objective where multiple states are defined, the data object $o \in O$ can have any of the specified states. A disjunction of the condition per state is added to the formal expression. The apartment *"Apartment 1"* in ω_3 is required to be either painted or finished.

$$\exists o \in O \,:\, o.class = Apartment \wedge o.id = \text{"Apartment 1"} \wedge (o.state = painted \vee o.state = finished)$$

For each link in the object diagram, a statement is created. It expresses that there exists a link $l \in L$, i.e., an unordered pair of data objects $o_1, o_2 \in O$. The data objects must belong to the

specified classes in the object diagram. If an identifier is given for the object nodes, the link must exist for the data object with the exact ID. If not, any object of the class is sufficient if it satisfies the specified states. The link between the house and *Apartment 1* in ω_3 is interpreted as follows:

$$\exists\{o_1, o_2\} \in L \ : o_1.class = House \wedge o_2.class = Apartment \wedge o_2.id = \text{"Apartment 1"}$$

In summary, for each data object node and for each link in the UML object diagram, a first-order logic statement is created. The objective is a conjunction of all these statements allowing for representing ω_3 as a first-order logic formula.

Objectives are first-order logic statements that are within the scope of the process model. The same holds for the presented formal representation of UML object diagrams as objectives. All formal expressions are made over the sets O and L of a case state. Given that the object diagram is consistent with the case model, the object diagram only expresses constraints about instances of data classes that are in the domain model. Also, it expresses conditions for the states of those instances that are part of the according object lifecycle. Furthermore, all links in the object diagram must be consistent with the domain model and the domain model describes all possible sets of links in any L. Therefore, all expressions generated for the links in the object model are in the scope of the state space as well.

4.2. Modeling goals with dependency graphs

A goal is defined as a combination of objectives over time. Reaching the objectives in a certain order allows for fulfilling the goal. We propose to utilize dependency graphs to model goals. Objectives are represented as nodes in the graph and their ordering is specified by their dependencies.

Notation. Goals consist of objectives and define ordering constraints over them. A dependency graph can be used to visualize this ordering. Dependency graphs are directed graphs and defined as a tuple $G = (N, E)$ of nodes and edges. For our purpose, the available nodes are a subset of all defined objectives $N \subset \Omega$. It must be finite and not empty. The edges are pairs of nodes, i.e., pairs of objectives, $E \subseteq N \times N$. In our context, dependency graphs must be acyclic to model goals.

Given the previously defined objectives ω_1, ω_2, ω_3, and a new objective $\omega_4 \equiv \exists o \in O : o.class = House \wedge o.state = finished$, which requires a house to be *finished*. A goal is a temporal ordering of those objectives. It can be denoted as a dependency graph. The goal γ_2 is one possible goal for the four objectives and is visualized in Figure 6. The dependency graph has a node for each considered objective $N = \{\omega_1, \omega_2, \omega_3, \omega_4\}$ and the set of edges is defined as $E = \{(\omega_1, \omega_2), (\omega_1, \omega_3), (\omega_2, \omega_4), (\omega_3, \omega_4)\}$.

Consistency with the data model. Like individual objectives, goals must be consistent with the domain model and the object lifecycles. The elements of the dependency graph itself are not bound to these models. Yet, the dependencies that are defined between objectives may violate the definitions in the process model. If the goal requires data objects to change from one objective to the next in a way that is not supported by the model, it is not consistent.

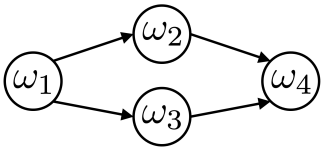

Figure 6: The dependency graph representing the goal γ_2.

First, the state changes of data object instances between objectives must be within the transitive closure of the object lifecycle. A data object instance can change its states only according to the object lifecycle of its corresponding data class. Therefore, two objectives must not require a data object to change its state in a way that is not supported by the object lifecycle. For instance, a goal might require the house *House1* to be *finished* and *started* in a later objective. The fCM does not allow such behavior and this goal can never be satisfied.

The links between data objects of two consecutive objectives must not contradict each other with respect to the domain model. For example, a first objective ω specifies that the apartment *Apartment1* is linked to the house *House1*. But in ω' *Apartment1* is required to be linked to the house *House2*. The domain model requires apartments to be linked to at most one house. Therefore, this goal would not be consistent with the domain model.

Formal representation. The dependency graph $G = (N, E)$ contains a set of objectives and a set of edges that describe the ordering of two objectives. To interpret these dependency graphs as goals, they can be formally represented as partial orders. A goal is a partial order of objectives such that each objective in the dependency graph is an element of the partial order. For all possible ordering constraints between two objectives in the dependency graph, the partial order orders the two.

Partial orders are transitive relations. Therefore, the dependency graph is interpreted as a transitive reduction of all relations in the partial order, which formally defines the goal. For each pair of objectives, where there exists a directed path in the dependency graph from the first to the latter, there is an ordering in the partial order. The example dependency graph depicted in Figure 6 can be formally represented as a partial order of its nodes representing objectives.

$$\gamma_2 = (\{\omega_1, \omega_2, \omega_3, \omega_4\}, \{(\omega_1, \omega_2), (\omega_1, \omega_3), (\omega_1, \omega_4), (\omega_2, \omega_4), (\omega_3, \omega_4)\})$$

For objective ω_1 there exists a path to every other objective. Therefore, the partial order contains a pair from ω_1 to every other objective.

Because the dependency graph is constrained to be acyclic, it always defines a finite set of sequences of ordered objectives. The partial order defines two possible sequences of objectives that are suitable to reach the goal: $\langle \omega_1, \omega_2, \omega_3, \omega_4 \rangle$ and $\langle \omega_1, \omega_3, \omega_2, \omega_4 \rangle$. The execution of the process model hast to allow satisfying at least one of the two sequences to satisfy the goal.

The proposed graphical goal models can be formally interpreted. Previous work shows that such formal goals can be used to automatically derive decision support for knowledge workers during run-time by [5, 22].

5. Prototypical implementation

We show how data-centric goals can be modeled visually as dependency graphs that order objectives, while objectives can be modeled as UML object diagrams. Showing the technical feasibility of the approach, we provide a proof-of-concept implementation[1] to test and use. The prototype extends the existing fCM modeling tool fCM-js [26], which offers design time support and automated guideline checking while creating fCM models. This tool is extended with visual modelers for creating objectives and goals.

The prototype allows for modeling multiple objectives and combining them into a goal. As previously elaborated, adherence to specific guidelines is imperative to maintain consistency across all diagrams. To address this requirement, we have incorporated interconnections between the modeler, the domain model, and the object lifecycles in our implementation.

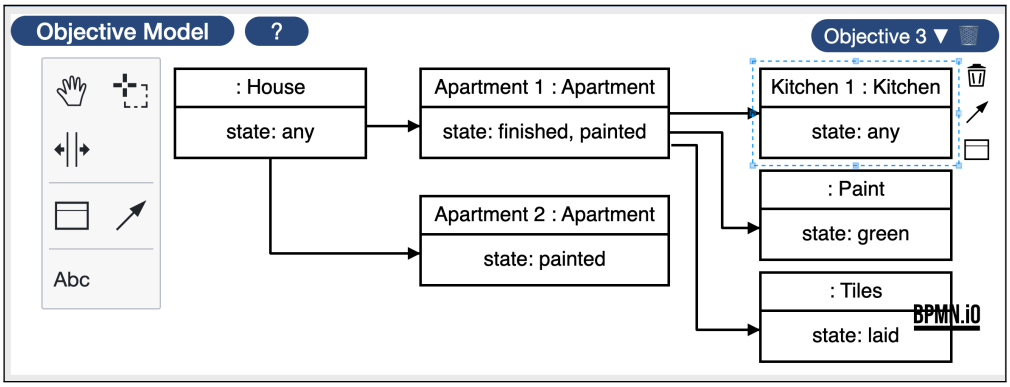

Figure 7: The objective ω_3 modeled in the objective modeler.

The objective modeler allows the modeling of UML object diagrams. In Figure 7, the previously introduced objective γ_3 is modeled in the prototype. As objects are derived instances of classes, users may select classes that are specified in the data model of the fCM when creating a new object. If a new class is created in the object modeler, it is automatically incorporated into the domain model. Similar functionality applies to the selection of states, which can be determined after class selection. The available selectable states are consistent with the corresponding object lifecycle associated with the referenced class. Additional to the object's states, the user may assign a name to each object, which remains exclusive to that particular object within the given objective. This allows referencing data objects across objectives.

Users can model multiple objectives that can be combined in the dependency modeler illustrated in Figure 8. It supports modeling a dependency graph of objectives that correspond to specific object diagrams in the objective modeler. Users can model the temporal ordering of the previously defined objectives by adding the desired dependencies between objectives.

Based on the defined goal model, the prototype can automatically generate a state space query, which can be used to derive recommendations [5]. Such recommendations return a Boolean for each next action indicating whether it allows reaching the goal.

[1]Source and Documentation https://github.com/bptlab/fCM-design-support/tree/state-space-queries

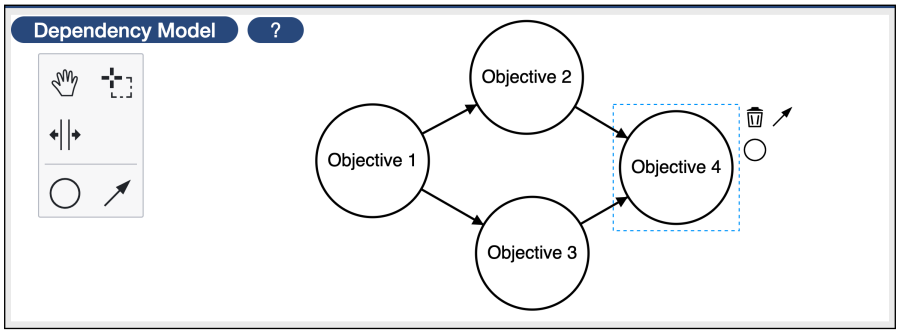

Figure 8: The goal γ_2 modeled in the dependency modeler.

6. Discussion and conclusion

Planning and goal modeling are essential in knowledge-intensive processes. Goals are a basis to provide decision and planning support for running process instances. We provide the means to model data-centric goals for fCM models graphically. Goals consist of objectives. Objectives express constraints over possible future execution states and can be modeled with UML object diagrams. Goals are a temporal ordering of objectives and can be defined as dependency graphs. We provide a formal interpretation of these models that can be utilized to derive decision support during run-time. Our approach is evaluated by a prototypical goal modeling tool.

The presented work has some limitations. In contrast to prior formal definitions of goals, UML object diagrams are less expressive than any first-order logic statement about a case state. Still, object diagrams are an established and well-known modeling technique and are more human-readable than first-order logic statements. In the presented object diagrams, attribute values in object diagrams are abstracted as data object states. In the future, a mapping of attribute values to data states could be beneficial for the practical usage of the presented goal models. The prototype is currently limited in its ability to derive decision support. Future implementations should extend the current functionality.

The benefit of the graphical goal models over formal expressions should be empirically evaluated in the future. Furthermore, the applicability as input for automated planning in BPM should be investigated. Also, the combination of the presented goal models with concepts of goal-oriented requirements language could allow for even more expressive goal models. Future work should investigate the application of the presented approach to different practical use cases resulting in domain-specific data-centric goal models.

The presented graphical goal models are based on UML object diagrams and can therefore be applied to any process model with a UML class diagram as a domain model like BAUML, OCBC, or BPMN. Consequently, the respective modeling techniques can now be extended with graphical goal modeling for running process instances.

References

[1] M. Weske, Business process management: concepts, languages, architectures, Springer, 2019.

[2] C. D. Ciccio, A. Marrella, A. Russo, Knowledge-intensive processes: Characteristics, requirements and analysis of contemporary approaches, J. Data Semant. 4 (2015) 29–57. doi:10.1007/s13740-014-0038-4.

[3] P. Pyöriä, The concept of knowledge work revisited, J. Knowl. Manag. 9 (2005) 116–127. doi:10.1108/13673270510602818.

[4] S. K. Venero, B. R. Schmerl, L. Montecchi, J. C. dos Reis, C. M. F. Rubira, Automated planning for supporting knowledge-intensive processes, in: Enterprise, Business-Process and Information Systems Modeling - 21st International Conference, BPMDS 2020, 25th International Conference, EMMSAD 2020, volume 387 of *Lecture Notes in Business Information Processing*, Springer, 2020, pp. 101–116. doi:10.1007/978-3-030-49418-6_7.

[5] A. Seidel, S. Haarmann, M. Weske, Model-based decision support for knowledge-intensive processes, Journal of Intelligent Information Systems (2022). doi:10.1007/s10844-022-00770-0.

[6] M. Estañol, Artifact-centric business process models in UML: specification and reasoning, in: R. Clarisó, H. Leopold, J. Mendling, W. M. P. van der Aalst, A. Kumar, B. T. Pentland, M. Weske (Eds.), Proceedings of the BPM Demo Track and BPM Dissertation Award co-located with 15th International Conference on Business Process Modeling (BPM 2017), volume 1920 of *CEUR Workshop Proceedings*, CEUR-WS.org, Aachen, 2017. URL: http://ceur-ws.org/Vol-1920/paper3.pdf.

[7] W. M. P. van der Aalst, A. Artale, M. Montali, S. Tritini, Object-centric behavioral constraints: Integrating data and declarative process modelling, in: A. Artale, B. Glimm, R. Kontchakov (Eds.), Proceedings of the 30th International Workshop on Description Logics, 2017, volume 1879 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2017. URL: https://ceur-ws.org/Vol-1879/paper51.pdf.

[8] S. K. Venero, J. C. dos Reis, L. Montecchi, C. M. F. Rubira, Towards a metamodel for supporting decisions in knowledge-intensive processes, in: C. Hung, G. A. Papadopoulos (Eds.), Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, SAC 2019, ACM, 2019, pp. 75–84. doi:10.1145/3297280.3297290.

[9] S. Steinau, A. Marrella, K. Andrews, F. Leotta, M. Mecella, M. Reichert, DALEC: a framework for the systematic evaluation of data-centric approaches to process management software, Softw. Syst. Model. 18 (2019) 2679–2716. doi:10.1007/s10270-018-0695-0.

[10] M. Hewelt, M. Weske, A hybrid approach for flexible case modeling and execution, in: M. L. Rosa, P. Loos, O. Pastor (Eds.), Business Process Management Forum - BPM Forum 2016, volume 260 of *Lecture Notes in Business Information Processing*, Springer, Cham, 2016, pp. 38–54. doi:10.1007/978-3-319-45468-9_3.

[11] S. Haarmann, WICKR: A Joint Semantics for Flexible Processes and Data, Ph.D. thesis, Universität Potsdam, 2022.

[12] E. Kavakli, P. Loucopoulos, Goal modeling in requirements engineering: Analysis and critique of current methods, in: J. Krogstie, T. A. Halpin, K. Siau (Eds.), Information Modeling Methods and Methodologies, Idea Group, 2005, pp. 102–124.

[13] D. Amyot, S. Ghanavati, J. Horkoff, G. Mussbacher, L. Peyton, E. S. K. Yu, Evaluating goal models within the goal-oriented requirement language, Int. J. Intell. Syst. 25 (2010) 841–877. doi:10.1002/int.20433.

[14] P. Kueng, P. Kawalek, Goal-based business process models: creation and evaluation, Bus. Process. Manag. J. 3 (1997) 17–38. doi:10.1108/14637159710161567.

[15] D. Greenwood, G. Rimassa, Autonomic goal-oriented business process management, in: Third international conference on autonomic and autonomous systems (ICAS'07), IEEE, 2007, pp. 43–43.

[16] D. Amyot, O. Akhigbe, M. Baslyman, S. Ghanavati, M. Ghasemi, J. Hassine, L. Lessard, G. Mussbacher, K. Shen, E. Yu, Combining goal modelling with business process modelling two decades of experience with the user requirements notation standard, Enterp. Model. Inf. Syst. Archit. Int. J. Concept. Model. 17 (2022). doi:10.18417/emisa.17.2.

[17] K. Decreus, M. Snoeck, G. Poels, Practical challenges for methods transforming i* goal models into business process models, in: RE 2009, 17th IEEE International Requirements Engineering Conference, IEEE Computer Society, 2009, pp. 15–23. doi:10.1109/RE.2009.25.

[18] G. Koliadis, A. Ghose, Relating business process models to goal-oriented requirements models in KAOS, in: Advances in Knowledge Acquisition and Management, Pacific Rim Knowledge Acquisition Workshop, PKAW 2006, volume 4303 of *Lecture Notes in Computer Science*, Springer, 2006, pp. 25–39. doi:10.1007/11961239_3.

[19] F. Dalpiaz, A. Borgida, J. Horkoff, J. Mylopoulos, Runtime goal models: Keynote, in: R. J. Wieringa, S. Nurcan, C. Rolland, J. Cavarero (Eds.), IEEE 7th International Conference on Research Challenges in Information Science, RCIS 2013, IEEE, 2013, pp. 1–11. doi:10.1109/RCIS.2013.6577674.

[20] S. Ghanavati, D. Amyot, L. Peyton, A systematic review of goal-oriented requirements management frameworks for business process compliance, in: Fourth International Workshop on Requirements Engineering and Law, RELAW 2011, IEEE Computer Society, 2011, pp. 25–34. doi:10.1109/RELAW.2011.6050270.

[21] A. Marrella, Automated planning for business process management, J. Data Semant. 8 (2019) 79–98. doi:10.1007/s13740-018-0096-0.

[22] S. Haarmann, A. Seidel, M. Weske, Modeling Objectives of Knowledge Workers, in: A. Marrella, B. Weber (Eds.), Business Process Management Workshops, Lecture Notes in Business Information Processing, Springer International Publishing, Cham, 2022, pp. 337–348. doi:10.1007/978-3-030-94343-1_26.

[23] Object Management Group, Unified modeling language (UML), 2017. URL: https://www.omg.org/spec/UML.

[24] H.-E. Eriksson, M. Penker, B. Lyons, D. Fado, UML 2 toolkit, John Wiley & Sons, 2003.

[25] H. Koç, A. M. Erdoğan, Y. Barjakly, S. Peker, Uml diagrams in software engineering research: a systematic literature review, in: Proceedings, volume 74, MDPI, 2021, p. 13.

[26] K. Andree, L. Bein, M. König, C. Mandel, M. Rosenau, C. Terboven, D. Bano, S. Haarmann, M. Weske, Design-time support for fragment-based case management, in: Business Process Management Workshops - BPM 2022 International Workshops, volume 460 of *Lecture Notes in Business Information Processing*, Springer, 2022, pp. 231–242. doi:10.1007/978-3-031-25383-6_17.