

Fundamental Conceptual Modeling Languages in OMiLAB

Dimitris Karagiannis, Robert Andrei Buchmann, Patrik Burzynski,
Ulrich Reimer and Michael Walch

Abstract Regardless of the application domain, both the analysis of existing systems and the creation of new systems benefit extensively from having the system modeled from a conceptual point of view in order to capture its behavioral, structural or semantic characteristics, while abstracting away irrelevant details. Depending on which relevant details are assimilated in the modeling language, modeling tools may support different degrees of *domain-specificity*. The boundaries of what *domain-specific* means are as ambiguous as the definition of a *domain*—it may be a business sector, a paradigm, or a narrow application area. However, some patterns and invariants are recurring across domains and this has led to the emergence of commonly used modeling languages that incorporate such fundamental concepts. This chapter focuses on the metamodeling approach for the hybridization of BPMN, ER, EPC, UML and Petri Nets within a single modeling method identified as FCML, with a proof of concept named Bee-Up implemented in OMiLAB.

Keywords Hybrid metamodeling • BPMN • ER • EPC • UML • Petri Nets

D. Karagiannis (✉) · P. Burzynski · M. Walch
Research Group Knowledge Engineering, University of Vienna,
1090 Vienna, Austria
e-mail: dk@dke.univie.ac.at

P. Burzynski
e-mail: patrik.burzynski@dke.univie.ac.at

M. Walch
e-mail: michael.walch@dke.univie.ac.at

R.A. Buchmann
Business Information Systems Department, Babes-Bolyai University,
400591 Cluj-Napoca, Romania
e-mail: robert.buchmann@econ.ubbcluj.ro

U. Reimer
Institute for Information and Process Management, University of Applied Sciences
St. Gallen, 9001 St. Gallen, Switzerland
e-mail: ulrich.reimer@fhsg.ch

1 Introduction

The goal of this chapter is to advocate the hybridization of widely adopted modeling languages. Thereby, the benefit is the availability of conceptualizations which have established foundations that can be specialized or extended in domain-specific modeling languages. The modeling languages under our scrutiny are BPMN [1], ER [2], EPC [3, 4], UML [5] and Petri Nets [6, 7]. Based on them, the FCML (Fundamental Conceptual Modeling Languages) modeling method was derived through a metamodeling approach that allows modeling with these languages within the same tool. The motivation behind FCML is manifold:

1. it is a multi-purpose method whose implementation enables users to model in several commonly used languages, in the same tool, thus defusing the typical decision dilemma in choosing, for example, which business process modeling language should be adopted in a certain enterprise; different modelers in an enterprise may require or have familiarity with different languages (e.g., CEOs preferring EPC, while CTOs favoring UML);
2. it exploits recurring semantics by allowing the user to execute certain mechanisms (e.g., simulations) on different notations that comply to specific patterns (e.g., workflow patterns); at the same time, it also provides language-specific mechanisms and language-independent mechanisms, by exploiting the different layers of abstraction involved in the hybridization of the different incorporated languages;
3. it opens possibilities for domain-specific extensions, semantic linking and lifting of what otherwise have been considered domain agnostic or general purpose languages.

For demonstration purposes, an academic proof of concept of didactic and experimentation interest was developed within the Open Models Laboratory [8] on the ADOxx metamodeling platform [9].

Additionally, the chapter discusses the metamodeling approach that is employed in the research environment of the Open Models Laboratory and therefore has enabled the works presented throughout this book.

The chapter is structured as follows: Sect. 2 will discuss the relation of the languages selected for the FCML method to domain-specific modeling and will clarify the OMiLAB assumptions about what domain-specific modeling is. Section 3 will provide background on the modeling languages assimilated under the FCML acronym and will establish the notion of modeling method and its metamodeling framework, as employed by FCML and also by the other OMiLAB projects. Section 4 will detail the FCML conceptualization in relation to the underlying platform's meta²model and Sect. 5 will showcase several key capabilities of the Bee-Up modeling tool, which implements FCML.

2 The Relevance of FCML for Domain-Specific Modeling

When designing languages for domain-specific modeling, a modeling method engineer will, on the one hand, (a) consider the established experience and lessons learned from standard languages or notations and, on the other hand, will (b) consider specializations and/or extensions with respect to *modeling requirements* raised for the addressed domain by the stakeholders who will either benefit from using models or work on the creation of models. Modeling requirements are commonly derived from two kinds of sources [10, 11]:

1. *Directly, from design-time needs* with respect to the capabilities of a required modeling tool. These typically pertain to the functionality that must support decisions regarding the engineering or re-engineering of a “system under study” (e.g., analysis, simulation and evaluation), to intrinsic qualities that models should have (e.g., understandability, semantic richness and consistency; see also existing frameworks for evaluating model quality [12, 13]) or to non-functional qualities that the modeling tool should have (e.g., usability, the ability to generate or reuse certain parts of models);
2. *Indirectly, from run-time needs* with respect to the capabilities of an information system that somehow makes use of the model contents—e.g., process-aware systems or other kinds of model-driven systems [14, 15]. The advocates of the model-driven engineering paradigm have emphasized the role of domain-specific modeling in capturing the domain concepts that are relevant to applications at run-time [16].

Such modeling requirements provide the starting motivation for the development of modeling languages with *domain-specificity*—that is, domain-specific modeling languages. The exact boundary of what “domain-specific” means, and where it differentiates from “cross-domain” or “general purpose”, is not fixed in an absolute way. Some languages are more specific than others, and some domains are narrower than others. The notion of *domain* itself may have different interpretations—it could be a business sector, a community-driven paradigm, a narrow application area or even a single (typically virtual) case of an enterprise that is not interested in model interoperability or understanding outside its environment. In this line of argumentation, we cannot argue that languages, such as those included in the proposed FCML method, are truly “general purpose” languages: UML is primarily involved in software engineering, compared to ER which has a narrower focus on data modeling; EPC and BPMN were designed for business process management and can be extended towards the more holistic scope of enterprise modeling. Petri Nets are the most abstract due to the fact that their inherent nature is based on a strong mathematical formalism, but their applicability is also clearly limited to a class of problems pertaining to process dynamics. Therefore, the languages discussed in this chapter, although addressing wider classes of problems than most of the methods described in this book, are also domain-specific in their own right, and some of them are more specific than the others—e.g., for describing a business process,

BPMN has more specificity than UML activity diagrams, as it will be stressed further in this chapter in an attempt to illustrate the generic-to-specific spectrum in the context of conceptual modeling.

The fundamental nature of a process, whose description may be traced back to ancient Greek philosophers and the “ontology of becoming” (and later to state-transition systems), is based on a flow that alternates transitions (changes, actions) with states (outcomes of changes, possibly considering also incidental external events). A conceptualization process led to translating this ontological view to *Transitions* and *Places* in Petri Nets, or to the more business-oriented *Functions* and *Events* in EPC. The reason why both exist, despite the obvious conceptual redundancy, is the different modeling requirements that they satisfy:

1. *on a syntactic level*: minimal notation in Petri Nets, to visualize some formalized behavior, versus color-coded and shape-coded notation in EPC to improve readability and cognitive effectiveness;
2. *on a semantic level*: formal semantics open to grounded interpretation (to enable cross-domain reuse) in Petri Nets, versus business concepts with non-local semantics, limited reuse, but good familiarity for targeted stakeholders, in EPC;
3. *on a functionality level*: focus on dynamic simulation and excitability in Petri Nets, versus focus on understandability and model interoperability in EPC.

Although approaches exist to cover all these classes of requirements, there is an inherent trade-off between machine-oriented executability and human-oriented understandability and this trade-off determines a polarization of requirements.

Modeling requirements also determine how we perceive the quality of models, as enabled by the modeling language. In an absolutist sense, “all models are wrong” [17], since all of them must leave out properties of the system under study (in this sense, domain-specific languages would be “less wrong” the more specific they are). Therefore, completeness, correctness, usefulness and other quality attributes should be judged in a frame that is built on the addressed requirements. While, for some users, model executability is essential (as input for some process automation system), for others, reasoning on model contents or cognitive effectiveness may be much more important. What some modelers would perceive as modeling agility, others may consider as ambiguous semantics. For exemplification, let us consider the following comparison:

- For some users, it is convenient to repurpose the UML activity diagram type as an algorithm flow chart notation in some contexts and for business process diagrams in other contexts. This is perceived as model agility since it allows a loose interpretation of the same notation, based on how the activities are named or based on some a priori understanding of what the model is expected to describe;
- Other users may require a clear distinction between high-level business tasks and low-level algorithm steps, between business decisions and conditional (IF) split nodes. Such a distinction imposes a more constrained use of the modeling tool/language, but also opens possibilities with respect to how models

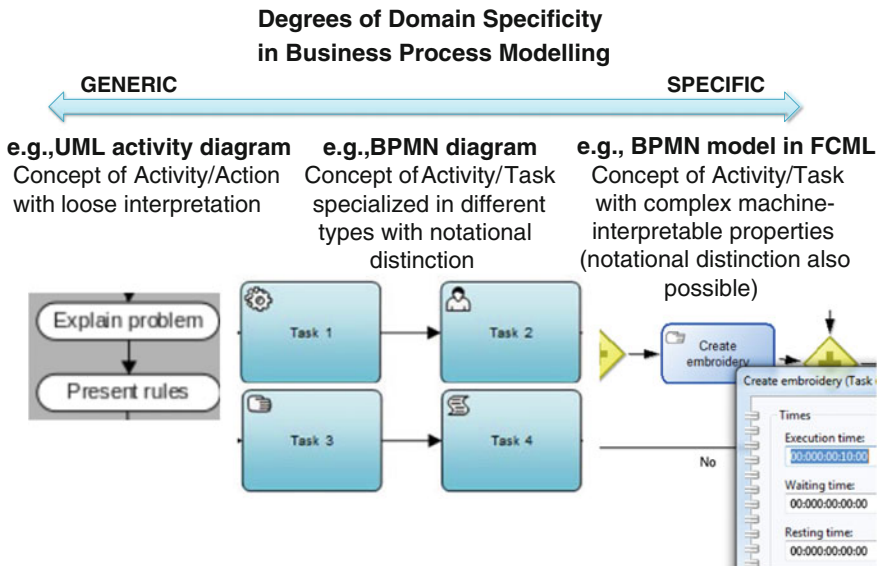


Fig. 1 Degrees of domain-specificity in business process modeling

can be processed by some model-driven functionality. The distinction may be enforced by the language syntax (e.g., subsumptions and notational variants for the same concept) or by the language semantics (i.e., explicitly defined in the language metamodel).

A straightforward example of the varying degrees of domain-specificity is illustrated in Fig. 1. A similar concept (*Activity/Task*) is presented with different notations and different semantics in UML activity diagrams, BPMN diagrams and FCML process models based on BPMN:

- In UML, the loose interpretation is possible by not fixing machine-interpretable semantics (except for the *Activity-Action* granularity distinction), but only a visual distinction from other types of nodes in the diagram. Domain-specificity is assimilated gradually as certain semantic aspects are fixed;
- BPMN adds typing (manual tasks, automated tasks, etc.) which is also reflected in the notation variability. This means that concepts from the application domain (here, business process management) become first-class citizens in the language alphabet, rather than being human interpretations of some generic symbols;
- Further on, the proposal of this chapter, FCML, adds property sheets to each *Activity* element, where the modeler may specify simulation-relevant attributes (e.g., different kinds of costs, times, resource consumptions) or semantic links (e.g., to a responsible role from a related organizational chart). These property sheets are prescribed by an “Activity schema” which is defined in the metamodel of the language as a means to provide semantics for the modeling

language constructs. Since the semantics is explicitly represented, it can be inspected and interpreted by the machine in order to impose a consistent model interpretation. The property sheet provides the definitorial attributes for the *Activity* concept in the context of this language: *an Activity is something that takes time, costs, must be performed by an organizational role with support from some enterprise resource*, etc. In a more general sense, such a concept schema may also be found in other knowledge representation approaches—e.g., formal concept analysis [18], ontology engineering [19, 20] and description logics [21]. The Semantic Web community works with such explicit, machine-interpretable semantics in order to achieve semantic interoperability across the Web and it also proposed applications for the metamodeling community [22]. In metamodeling, such a description is implemented on the underlying metamodeling platform and makes it impossible to interpret a business process as an algorithm flow chart. The freedom of interpretation is thus traded for a richness of semantics on which business-oriented functionality may be built—e.g., simulation of different properties with different kinds of meaningful aggregations (total costs, lists of employees involved on a process path, etc.), cross-model queries for enterprise analysis, etc.

FCML extends both UML and BPMN activities with domain-specific properties that specialize their semantics in an enterprise modeling context. In addition to attributes like costs and times, Fig. 2 shows an example where a BPMN task is assigned to its responsible performer/role not only through the visual means that BPMN provides (e.g., containment in a swimlane/pool) but also through a machine-interpretable semantic link to the organizational chart, as modeled within its own context (an organizational structure model with departments, performers, roles, etc. which may have its own domain-specific elements or editable properties).

These examples show how a modeling language may include concepts of varying domain-specificity even within the same model, or across different implementations. The challenges identified by the paradigm of “multilevel modeling”—see [23] further refine this aspect and contribute to a more flexible view on what the boundaries of a modeling language are. Although a scale from generic to specific may emerge from this discussion (as shown in Fig. 1), it will not hold in the general case. If we add Petri Nets to the discussion, their positioning in the spectrum is unclear (activity diagrams do not have a specialization of the *Place* concept, whereas in FCML the UML activity may have domain-specific semantics).

The goal of this section was to clarify our interpretation on domain-specificity in relation to FCML and with respect to the scope of this book. On the one hand, we tried to defuse the overly simplified traditional dichotomy between *general purpose* and *domain specific* languages, at least in the context of conceptual modeling. On the other hand, we aimed to remove also the simplification that modeling languages should be positioned in a linear range from generic to specific, as they employ different conceptual constructs. Consequently, we assigned the moniker *Fundamental Conceptual Modeling Languages* to the languages selected for hybridization under FCML, due to their quality of established references and starting points for

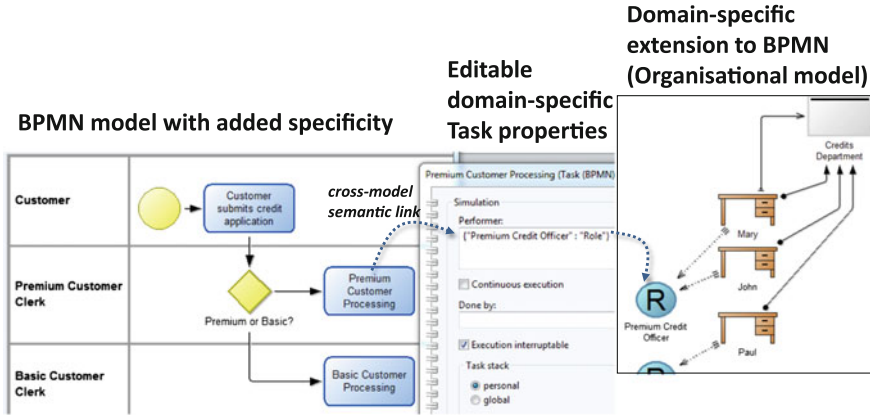


Fig. 2 Extending the domain-specificity of BPMN concepts in the FCML implementation

the concept specialization that is typically required in domain-specific modeling languages. The message of this section is that, when dealing with knowledge representation, any generic-to-specific variation should be discussed on concept level rather than language level, and this conclusion is aimed at extending the previously stated motivation behind FCML.

3 Method Description

3.1 Background on the Fundamental Conceptual Modeling Languages

The FCML modeling method incorporates and extends several modeling languages that gained wide popularity and are supported by communities with the help of a wide array of modeling tools, both commercial and free. This section provides a brief overview on the assimilated languages, to be later illustrated also by the proof of concept implemented in OMiLAB.

Entity-relationship (ER) diagrams have been widely adopted in the conceptual modeling community as the fundamental approach for data modeling, starting with the milestone paper of Peter Chen [2]. ER models have an ontological nature, in the sense that they describe categories of being and their relations, thus having a scope similar to that of UML class diagrams or metamodels. However, the objectives of ER modeling have been traditionally related to data modeling and database design, a prominent use being the generation of data schemata [24] or reverse engineering diagram generation [25]—typically for relational databases, but not necessarily [26, 27]. The ER metamodel is highly abstract, dealing with *Entities* rather than “paradigm-specific” tables/tuples. Therefore, ER diagrams may also describe data

intended to be stored in other data structures. Its core concepts are the *Entity*, the *Relationship* (that exists between Entities) and the *Attribute* (of an Entity or Relationship). Additional properties of these are the primary key (for *Entities*), cardinalities and roles (for *Relationships*). Several extensions have been proposed over time—e.g., the “extended” ER (E^2R) adds subsumption, thus allowing for entity specialization [28, 29]. The typical usage of ER diagrams is in the requirements analysis and design phases when the modeler employs ER to refine granularity and to adapt a data model across the conceptual-logical-physical layers. One of the mechanisms typically associated with ER diagrams is the generation of database schemata, for example by deriving SQL statements that are on the same level of abstraction and detail as the diagram content. A flagship conference [30] became the forum of a community that initially revolved around concerns related to ER modeling, later expanding according to the different “waves” of modeling approaches developed over decades, including the one driven by the standardized Object Management Group languages such as UML [5].

Unified Modeling Language (UML) is one of the most prominent standards in software engineering, a language established in the late 1990s to support a unified method for object-oriented software development, by incorporating lessons learned from the large number of modeling languages that had been in use during the 1980s and early 1990s [31]. Therefore, UML may be seen as a natural descendant of the simpler and more focussed ER modeling approach. It covers a much wider scope through a number of diagram types addressing various aspects of a software system, classified into two categories: *static* (structural)—e.g., class diagrams, component diagrams and *dynamic* (behavioral)—e.g., activity diagrams and sequence diagrams. It still shares with ER the desideratum of code generation; however, UML addresses an object-oriented development context (e.g., class definitions derived from class diagrams). Additionally, UML fuelled the model-driven software engineering paradigm, due to some key strengths that are complementary to the modeling language itself: (a) model interoperability through diagram interchange formats—XMI [32]; and (b) a standard constraint definition language—OCL [33]. The notions of *UML profiles* and *stereotypes* were introduced to enable customization of the language alphabet for different development paradigms—e.g., XML-based applications [34]; or even domain-specific extensions—e.g., SysML [35], SoC [36]. Just like ER modeling, UML also ignited research interests and a community aggregated around a long-standing scientific conference—MODELS [37].

Business Process Model and Notation (BPMN) is an OMG standard [1] designed to support the business process management paradigm with a more extensive range of diagrammatic possibilities compared to traditional flowcharting or UML activity diagrams. One of the key benefits of BPMN is the domain-specificity added by typing generic concepts that have been available in traditional flowcharting languages. This specificity (addressing the “domain” of business process management) manifests as a richness of types (*Task* types, *Event* types, *Gateway* types) that provide semantic enrichment for not only human-readable interpretation but also for executability—thus stimulating the rise of business process execution engines. This was possible in tandem with the

syntactic interoperability means provided by the XML ecosystem—specifically, the dedicated schemata for capturing a machine-processable serialization of diagrammatic process descriptions: BPEL [38], XPD [39]. BPMN places a strong focus on the notational level, with the semantic variability being reflected in notational variability, through visual cues added to the shapes that represent tasks, events, gateways, etc.; at the same time, translations between BPMN and BPEL have been proposed [40] to support executability. Limitations of such mappings have been discussed in the sense of a conceptual mismatching between the diagramming standard and the serialization standard [41]. Trade-offs must be made between understandability and the formal rigour required for process executions, and consequently, subsets of the modeling constructs have been proposed in BPMN 2.0, addressing different modeling scopes—see also the analysis of [42]. The overall scope of BPMN being limited to business process descriptions, it provides only minimal support for describing the enterprise context—e.g., swimlanes reflecting organizational responsibilities for different parts of a process. Decision logic was recently separated from BPMN in a complementary modeling language—Decision Model and Notation [43].

Event-driven Process Chain (EPC) diagrams were introduced by the framework of Architecture of Integrated Information Systems (ARIS) and its software tools [3, 4]. EPC shares with BPMN the targeted domain (business process management), although the exact scope is different, due to a different trade-off between understandability and underlying formal rigour and due to how they are contextualized in enterprise architectures—see a comparative analysis at [44]. EPC advocates cognitive effectiveness through color coding and shape coding while removing the rich taxonomical classifications promoted by BPMN (perceived as excessively complicated by certain stakeholders, see [45, 46]). However, EPC formal semantics have been analyzed (with the help of Petri Nets in [47]) and serializations for model-driven systems have been proposed—ARIS was an early adopter of XPD, attempts at BPEL serialization have been discussed [48] and an EPC-specific XML vocabulary has been proposed [49]. The core concepts of EPC are the *Function* and the *Event*, which can be interpreted as a flow of alternating “changes” and “states”, with *Functions* being connected to elements of enterprise context: responsible organization unit, supporting IT system, input and output information. EPC shares with BPMN the basic control flow split and merge nodes of different logical types (XOR, AND and OR). Unlike BPMN, it stresses the need to identify states that emerge from, or trigger the need for the execution of *Functions* (events also exist in BPMN, but quite often they are considered implicit between two consecutive tasks). Another important distinction in general use is that EPC adds elements of the enterprise context (e.g., organizational units responsible for performing a *Function* are a language concept, not only a visual container). Actually, EPC emerged from an integrated way of modeling enterprise architecture, which is out of scope for BPMN.

Petri Nets [6, 7] is one of the longest standing diagrammatic modeling methods, with minimal but powerful semantics based on strong mathematical foundations. The trade-off here is that, on one hand, the method is sufficiently abstract to have

cross-domain applicability with respect to process dynamics (especially relevant in the context of distributed systems); on the other hand, the level of abstraction imposes a learning curve that is not typically acceptable for business stakeholders and consequently the method was developed rather for academic concerns or as an underlying abstraction for other languages—see the effort of defining Petri Nets-based semantics for UML activity diagrams [50] or EPC models [47]. The Petri Nets method has a minimal metamodel that includes three highly abstract concepts—*Places* (states), *Transitions* (changes, actions) and *Arcs* (indicating the flow of places/transitions). The behavioral dynamics of a system are captured by a property called *Token*, which may be passed along the flow of places, each passing being triggered by the *firing* of a transition, which signifies an action taken. Simulation mechanisms are employed to monitor the possible states of the system as a whole, based on how the transitions may be fired, on how multiple possibly concurrent tokens are passed along, as well as on token availability in different places during different system states (token availability in a selected place will enable the transitions following that place). Typical simulation goals are the assessment of reachability of certain states, the risk of deadlock, the liveness/deadness of certain transitions. Extensions such as Colored Petri Nets [51] or transfer/reset nets [52] were introduced to enrich expressivity with additional properties (e.g., guarded transitions, token data, reset arcs).

3.2 The Metamodeling Approach

The engineering of a hybrid modeling method such as FCML must follow a *metamodeling approach* to ensure proper semantic coverage, and to ensure that the method is not only bringing different types of diagrams in the same modeling tool, but it also adds the following benefits:

- It extends *diagrams* to the status of *conceptual models* in the following sense: the model is not only a notational construct built with different graphical shapes relying on human interpretation; instead, each shape is instantiated from a higher abstraction concept with explicitly defined, machine-interpretable semantics, based on a concept taxonomy and descriptive properties through which the language terminology is defined. This allows both structural and semantics-driven processing of models, including reasoning on the structure or domain-specific properties of model elements, with rules processed by means that are specific to each metamodeling platform (the FCML implementation in OMiLAB was built on ADOxx [9]). The main distinction from other knowledge representation paradigms (e.g., description logics, ontology engineering) is that with metamodeling there was less effort towards interoperability across the popular metamodeling platforms—EMF [53], ConceptBase [54], MetaEdit + [55, 56]—whereas description logics and ontology engineering are following a trend towards the unifying logic envisioned in the Semantic Web “layer cake”

- and a Web-oriented standardization overlooked by the W3C through several drivers—e.g., RDF for representing facts [57], OWL as an ontology language [20], RIF as a rule interchange format [58]. To stimulate the emergence of a similar unifying abstraction layer, OMiLAB has initiated a cross-platform language for modeling method definition, MM-DSL, with an early draft discussed in [59];
- Furthermore, the metamodeling approach integrates models with meaningful cross-model relations that will act on one hand as semantic relations between concepts of different languages and on the other hand as hyperlinks that support navigation across models (thus improving usability and understandability). This will be showcased in Sect. 5 as a means of making models compliant to cross-notational simulation algorithms, but is not limited to this.

Figure 3 depicts the abstraction layers involved in the metamodeling approach of OMiLAB. A standardized version of a multi-layered conceptual architecture was also defined as a framework for UML, under the name MetaObject Facility [60]. We provide here a brief explanation on these layers:

- On the *Modeling layer*, models are created according to a specific modeling language, with distinctive notation and semantics for each diagrammatic symbol;
- To make modeling possible, on the *Meta layer* the terminology of the language is prescribed by specifying: the concepts that are allowed to be used, their notational manifestations, their semantics (property-based descriptions),

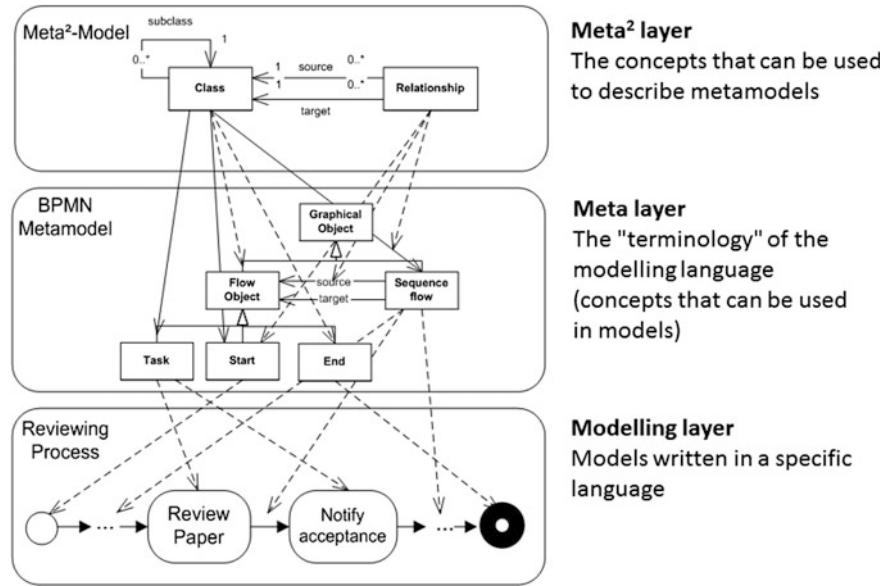


Fig. 3 Abstraction layers in metamodeling—adapted from [65]

syntactic and semantic constraints (e.g., domain or ranges for visual connectors). Without these, a modeling tool is a semantically agnostic diagramming tool. This is where the metamodeling effort and hybridization take place primarily and the result is a terminological structure extended with dynamic visualization and all the properties that are necessary for the required modeling functionality (e.g., simulation);

- The creation of the language terminology requires itself a *Meta² layer*, where several foundational meta-concepts (e.g., class, property) provide invariants that are instantiated in language metamodels. Metamodeling platforms provide these pre-defined invariants: some popular examples are ADOxx [9], GOPRR [61], ECore [62]—see an overview in [63]. Within our metamodeling approach, we investigated current metamodeling approaches and their meta-concepts and proposed meta-concept extensions on Meta² layer for systematic modularization and flexible composition of metamodels, an important aspect of the engineering of hybrid modeling methods [64].

The result of the OMILAB metamodeling approach is encompassed by the notion of *modeling method*, which extends that of a *modeling language*. The modeling method was defined in [65] and its building blocks are depicted in Fig. 4, with a possible formal view provided in [66]:

1. The *modeling language* provides the set of modeling constructs (their notation, grammar and semantics). The language grammar (syntax) defines fundamental modeling constructs of the language and relationships between them, whereas the notation is concerned with the form of the language [67]. By assigning visual representations to modeling constructs, the semantics (static and dynamic) specifies unambiguously the meaning of language constructs [68]. To achieve manageable granularity and understandability, the language may be

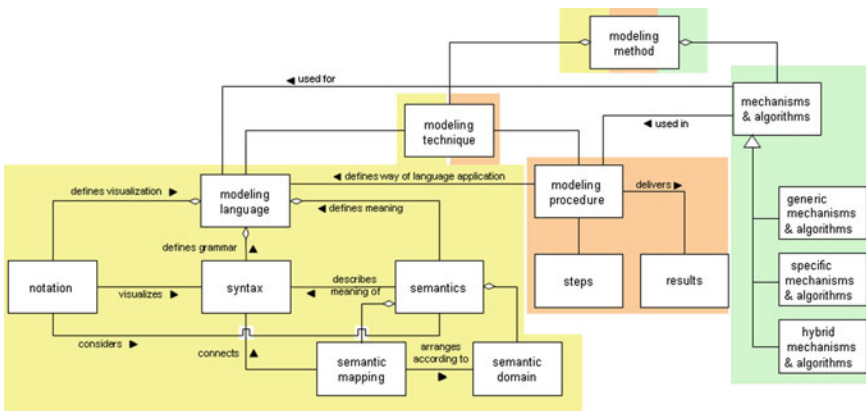


Fig. 4 The building blocks of a modeling method [65]

partitioned in *model types* addressing different facets or dimensions of the modeled system. This partitioning can be motivated as a usability feature (a top-down decomposition approach to avoid visual cluttering) or as a consequence of hybridization (a bottom-up strategy employed to interconnect modeling language fragments). In any case, the model types can be connected with semantic hyperlinks that enable cross-model navigation, as well as cross-model functionality. The work at hand exploits this by semantically extending generic concepts with additional properties that relate them to domain-specific concepts in other model types (e.g., see in Sect. 5 the case of the EPC extension);

2. The *modeling procedure* defines the steps that must be taken by modelers towards their goal. In the simplest case, it advises on the precedence in creating different types of models in order to achieve a coherent set of related models of different types. This includes the preparation of cross-model links to enable specific functionality that relies on certain model patterns, to be highlighted in Sect. 5;
3. The *mechanisms and algorithms* cover functionality that must process model contents for various purposes (simulation, visualization, transformation, evaluation, etc.). In this respect, the work at hand illustrates the generic-to-specific spectrum with respect to modeling functionality:
 - (i) The *SQL generation mechanism* is specific to ER modeling and *token-based simulations* are specific to Petri Nets;
 - (ii) Workflow simulation mechanisms are less specific, as they apply to a wide range of model types, as long as they allow the description of the typical workflow patterns. *Path analysis* or *workload assessment* may aggregate domain-specific properties attached to the control flow model types (BPMN, EPC and UML activity diagrams). Similarly, *reasoning mechanisms* may apply to several model types. For example, querying conceptual models might require reasoning (e.g., to account for inheritance) and can be defined for models represented in UML class diagrams as well as for ER models. Reasoning mechanisms can also provide *inference services* to be utilized by an application that is generated from the model or built on top of it, e.g., for classifying instances into classes. Furthermore, reasoning mechanisms enable *consistency checking* of a model during build time to support the modeler in creating and editing a model [69]. The means for implementing simulation and reasoning mechanisms, as well as other algorithms associated with a modeling language, would be highly dependent on the metamodeling platform. In this respect, ADOxx provides the AdoScript scripting language which can be used for programmatic implementations driven by the machine-interpretable model semantics (e.g., the graph rewriting mechanism described in [70]);

- (iii) The algorithms that come along with a modeling method (such as the ones mentioned above for generating SQL statements, for simulation, and reasoning mechanisms in general) interpret the constructs of the underlying modeling language in a specific way in order to implement the intended functionality. In other words, the algorithms attribute a specific meaning or semantics to the constructs. Instead of having the semantics local to the algorithms, which introduces the risk of inconsistent interpretations between different algorithms, it is preferable to represent the semantics in an explicit way as part of the modeling language, i.e., in the metamodel. The Semantic Web community achieves such a *machine-interpretable semantics* by formally grounding their modeling languages in description logics [21]. Machine-interpretable semantics opens up new possibilities, such as (semantic) interoperability and (semantic) bridging between models. An example of model mapping, which benefits from such a machine-interpretable semantics is illustrated in Chapter 19 of this book (formally described in [71]), where models regardless of their type are *converted to Linked Data graphs to allow reasoning on model contents* through standardized means using the RDF framework [57]. The metamodeling community has so far been less interested in semantic interoperability across metamodeling platforms and the resulting opportunities—therefore, we mention this challenge here as a current research opportunity.

Based on the definition discussed here, a domain-specific modeling tool must implement a complete method and not only a language. Consequently, the tool should include (a) model-driven functionality that is relevant with respect to the modeling requirements; (b) guidelines and constraints for modeling scenarios with respect to different modeling goals and related functionality. The next sections will emphasize this aspect by using the case of FCML and its Bee-Up implementation.

4 Method Conceptualization: The Underlying Meta²model

To achieve its hybridization goal, FCML makes use of the meta²model foundational constructs provided by the ADOxx metamodeling platform [9] whose meta²model was analyzed in detail in [72]. The choice of platform and the implementation followed certain principles, such as: minimizing the workarounds, having the platform-specific metamodels as close as possible to the original ones, having the possibility of restricting metamodels through configuration rather than because of platform restrictions. The ADOxx meta²model constructs are shown in Fig. 5 and a mapping of their relevance for each language assimilated in FCML is provided in Table 1.

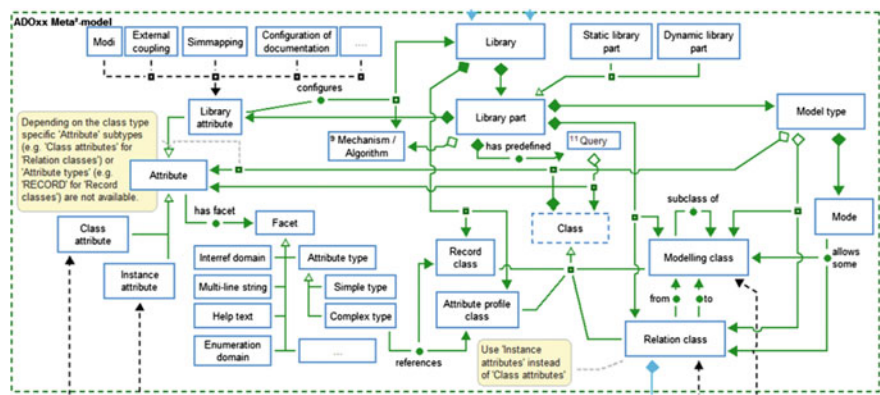


Fig. 5 The underlying ADOxx platform meta²model

Table 1 Involvement of different ADOxx concepts in the FCML components

Meta²-model Concept	BPMN	EPC	ER	UML	PN
Library	✓	✓	✓	✓	✓
Static Library	≈	≈		≈	
Dynamic Library	✓	✓	✓	✓	✓
Model Type	✓	✓	✓	✓	✓
Mode	✓	✓		✓	
Modelling Class	✓	✓	✓	✓	✓
Relation Class	✓	✓	✓	✓	✓
Record Class				✓	
Attribute Profile Class					
Attribute (Not Platform-specific)	✓	✓	✓	✓	✓
Class Attribute					
Instance Attribute	✓	✓	✓	✓	✓
Inheritance (subclass of)	✓	✓	✓	✓	
Query (Platform-specific)				✓	
Simulation (Platform-specific)	✓	✓		✓	
Custom Algorithms/Mechanisms	≈	≈	✓	≈	✓

- A *Library* contains the ADOxx definition for a modeling method, including its language definition, mechanisms and algorithms. It typically has two parts: the *Static* part covering structural model types (e.g., UML class diagrams, ER models, organizational charts, etc.) and the *Dynamic* part covering behavioral model types (e.g., UML activity diagrams, BPMN, EPC and Petri Nets). The table suggests that the Static part is not inherent to BPMN and EPC, but was added as domain-specific extensions in FCML. Several *Library attributes* act as metadata with possible coupling to external systems (e.g., external scripting or system commands);

- A *Model Type*, as already explained in the previous section, is a partition of the hybrid modeling language alphabet, thus serving a separation of concerns and including only the concepts that are relevant to a particular aspect of the system under study. FCML provides a model type for each of the UML diagram types, one for BPMN, one for EPC, one for ER, one for Petri Nets and additional ones as domain-specific extensions (e.g., organizational “work environment” models);
- A *Mode* is a subset of a Model Type which restricts the use to a limited set of constructs determined by frequency of use or functionality requirements for a particular class of problems (e.g., a simulate-able model might require more concepts compared to a model created strictly for human communication);
- A *Modeling Class* is a metamodel concept that can be instantiated in models in the form of a directed graph node. Such a concept is defined in terms of (a) a “schema” prescribing its set of definitorial and descriptive properties and (b) a notation that can be customized according to required visual dynamics (e.g., interaction points as functionality triggers, notation variability determined by instance-level property values);
- A *Relation Class* is another type of metamodel concept, a connecting concept that can be instantiated in models in the form of visual connectors. Connectors have their own schema with their own properties and possibilities for notational customization—however, constraints such as domain, range, cardinality and relational notation must be considered;
- A *Record Class* is a schema for a tabular property that may be included in the prescribed property sets of any modeling concept, to collect property values that are complex and cannot fit a simple property slot. The use of this kind of properties is limited in FCML to only a few of the UML constructs (e.g., class attributes in a class diagram);
- An *Attribute Profile* is a schema for a set of properties that should be reused throughout the metamodel but will not act itself as the schema of a modeling concept. Currently, FCML does not employ this component;
- An *Attribute* is a property attached to the semantic definition of a modeling concept. They can be made visible as editable attributes, in property sheets that are attached to any model element, or they can be used strictly for inheritance purposes at metamodel level. Their *Facets* allow for additional restrictions on their value range, including the possibility of having links (*interrefs*) to elements from different model types;
- The *Inheritance* indicates the possibility of reusing modeling concepts by inheriting their “schema” in more specialized concepts. This is one of the key enablers for extending concept specificity in domain-specific modeling languages and has been extensively used in FCML, with the exception of Petri Nets whose minimal metamodel does not require inheritance;

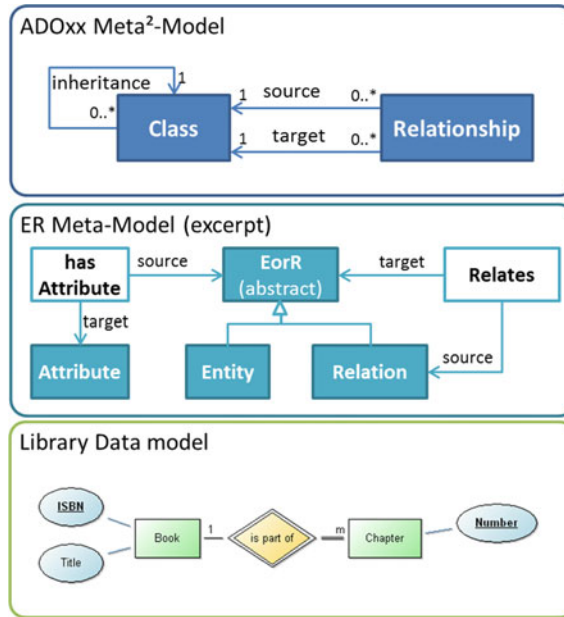


Fig. 6 The ER metamodel assimilated in FCML

- The *Query* indicates the possibility of inheriting an internal query engine built on the generic structure of any model, as well as the customization of pre-defined queries for each specific model type. In the current implementation, FCML does not fully employ this component, although work is underway to design model queries that are relevant to the specificity of each model type;
- The *Simulation* indicates the possibility of inheriting several simulation engines that are applicable on models of various specificities if they comply with certain modeling patterns (e.g., workflow patterns in BPMN, EPC, UML activity diagrams). Section 5 will illustrate how this applies to FCML;
- The *Custom Algorithms/Mechanisms* indicates the possibility of extending model-driven functionality with any customized mechanism (e.g., reasoning, code generation) based on the specific semantics and structure of each model type. These can be programmed in the internal AdoScript language and Sect. 5 will illustrate such possibilities for ER and Petri Nets. The other FCML model types can also make use of the customized mechanism of a more generic nature (e.g., model exporting).

Figures 6, 7 and 8 provide a more detailed view on the core meta²model concepts that are involved in the metamodel for each of the languages assimilated in the FCML (ER, BPMN, EPC, UML—partially depicted, and Petri Nets, respectively).

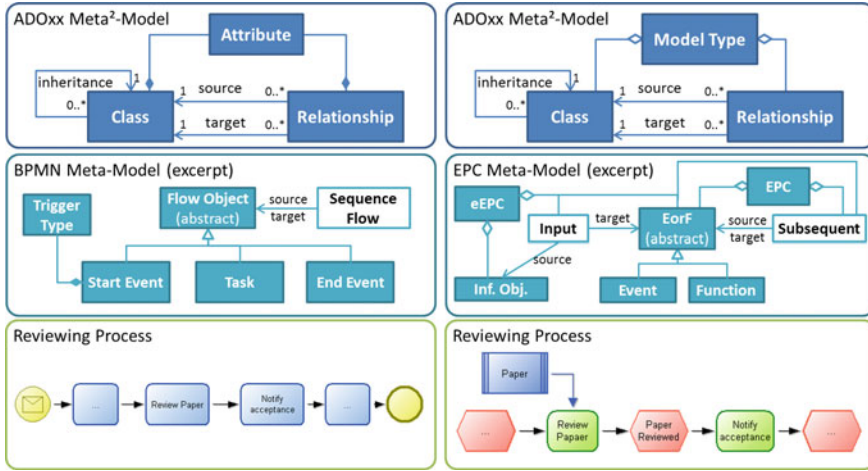


Fig. 7 The BPMN and EPC metamodels assimilated in FCML

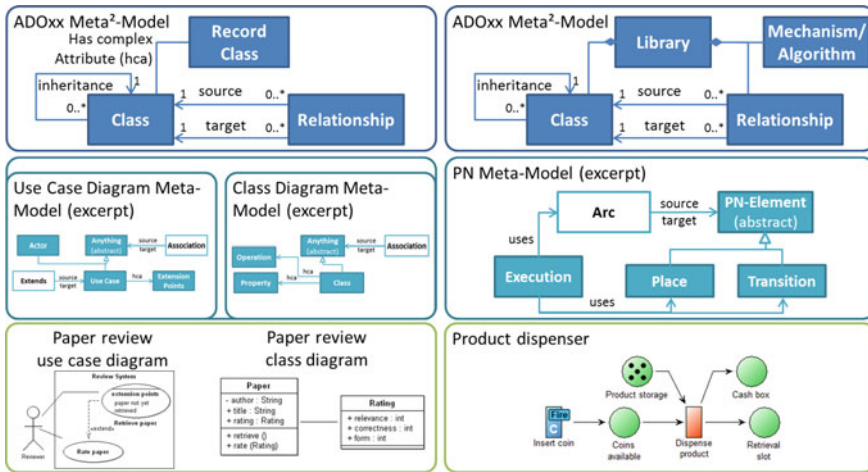


Fig. 8 The UML (fragment) and the Petri nets metamodels assimilated in FCML

5 Proof of Concept: The Bee-up Tool

A proof of concept for FCML was implemented in the OMiLAB environment as the Bee-Up modeling tool [8]. We take the opportunity to showcase in this section how the implementation extends the original specifications of the assimilated languages, due to the added specificity as well as due to method-level integration on the meta²model foundation provided by ADOxx.

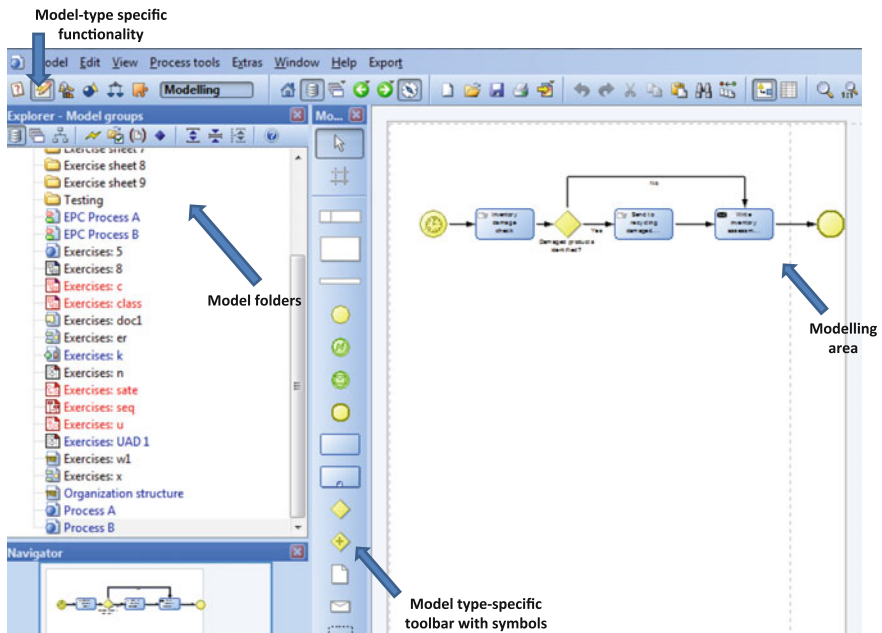


Fig. 9 The Bee-Up user interface

Figure 9 shows an overview of the Bee-Up user interface, with its model management component, the main menu providing implementations of FCML mechanisms and the modeling area providing a Model-Constructs ribbon that is specific for each type of model (determined by its metamodel).

The creation of a model will trigger a panel where the FCML languages are classified according to the categories that may also be found in UML (static or dynamic)—however, we include here all the FCML model types (notice that EPC and BPMN span across the two categories with elements of domain-specificity that are relevant, for example, to simulation mechanisms).

Figure 10 shows this panel and the list of UML model types, as well as several samples of UML models (sequence, class and state diagram). Each diagram element (nodes, containers, connectors) has a sheet of editable properties (bottom-left side of Fig. 10), which includes the possibility of extending the semantic of all the concepts found in the FCML languages with domain-specific properties, additional typing or semantic hyperlinks to related models.

Another model type is Petri Nets. The official notation of Petri Nets is quite minimal, as most languages that have not been designed with enterprise modeling requirements in mind (it provides only the minimal distinctions necessary to grasp

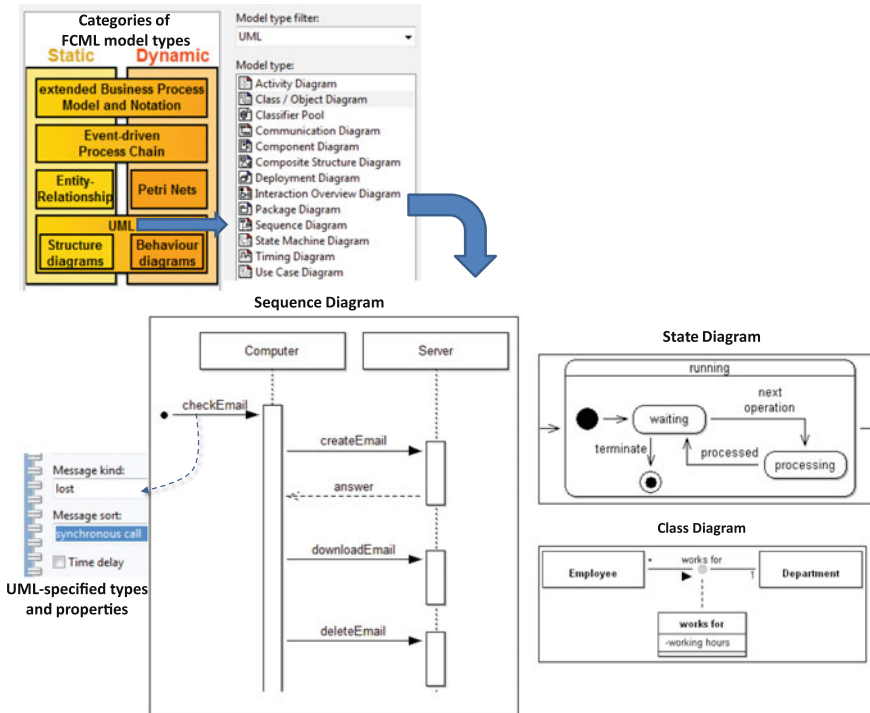


Fig. 10 The UML model types included in FCML

the underlying mathematical formalism). The Bee-Up implementation adds on the notational level several visual cues and visual dynamics to facilitate user interaction in support of Petri Nets-specific simulation mechanisms. Figure 11 shows: (a) interactive visual cues (the *Fire* boxes) that may be used by the modeler to step through the states of the model; (b) a purely functional symbol added to the modeling language to provide simulation triggers with preset parameters (e.g., transition priorities) directly on the modeling canvas; (c) symbols that may store and restore relevant system states described in terms of the number of tokens present in each place.

Figure 12 shows an example of a BPMN process model together with simulated costs for a particular process path (highlighted by the notational dynamics that can be programmed in ADOxx). This is actually a *Path analysis* mechanism that is domain-specific in the following senses: (a) it is applicable only to those models whose structure conforms to workflow-specific patterns (e.g., BPMN, EPC and UML activity diagrams); (b) it aggregates domain-specific properties that were

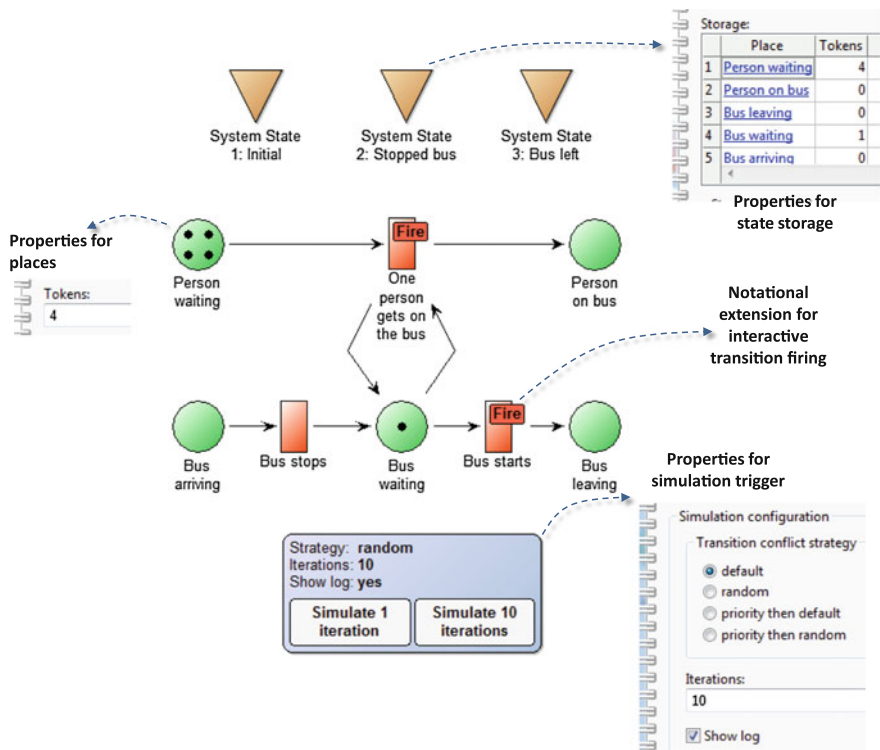


Fig. 11 Petri Nets simulation controlled by interactive visual cues

attached to the process steps along each path (e.g., different kinds of costs, times, domain-specific resource consumptions); these extensions are applied on the metamodel level for the corresponding *Task/Activity* concept in each model type, inheriting some of them from a higher level concept that acts as the hybridization bridge of FCML.

Figure 13 illustrates another domain-specific simulation mechanism, this time applied to EPC models which were semantically extended at metamodel level with hyperlinks between EPC functions and roles or performers from a distinct organizational model. The existence of a separate organizational structure model type allows multiple EPC processes to be linked to the same human resources or responsible units while at the same time avoiding the visual cluttering that would occur by reusing the organizational unit on the modeling canvas of each EPC model. Besides the domain-specific aspects that can be added to the organizational units without affecting the EPC metamodel, the key benefit of this “separation of

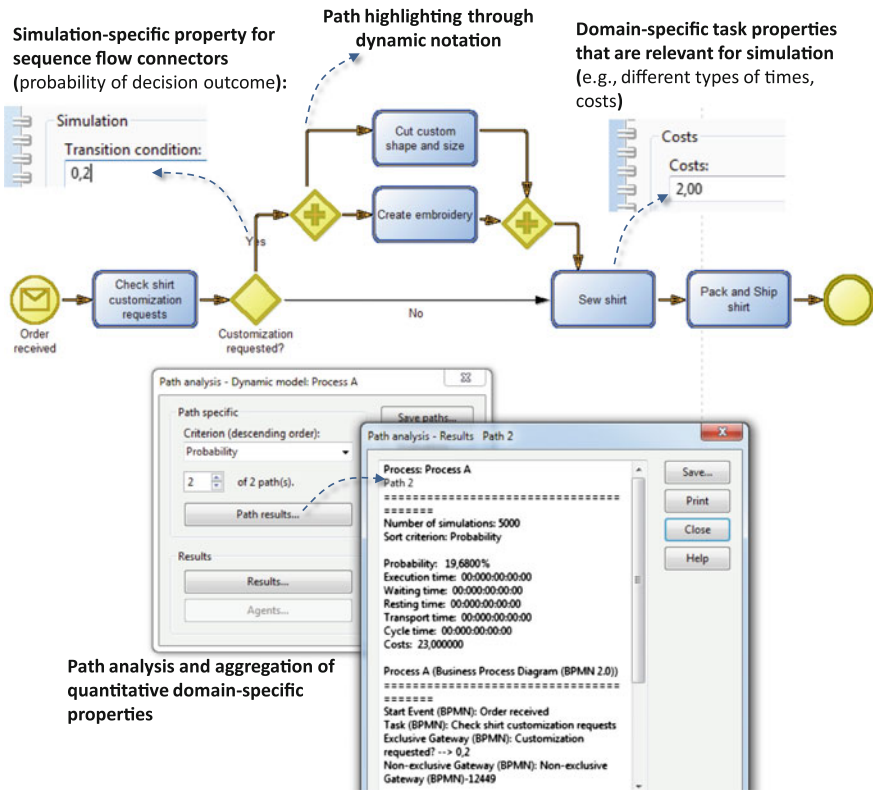


Fig. 12 Path analysis on BPMN model extended with domain-specific task properties

concerns” is the ability to build on it the workload assessment simulation that aggregates relevant properties (e.g., work times) for the organizational unit under scrutiny, as suggested in Fig. 13.

Figure 14 shows an example of a simple ER model and its key mechanism—SQL code generation. The editable property sheets which were used in previous examples to collect domain-specific properties (e.g., for enabling simulation) were tailored here to capture information that is specifically needed for the SQL generation mechanism (e.g., data types or key options for attributes, roles or cardinalities for entity–relationship arcs).

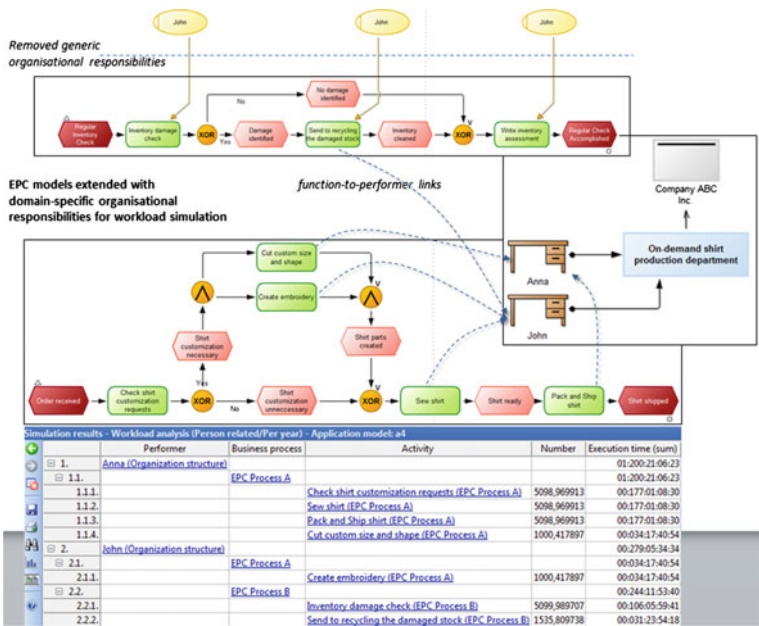


Fig. 13 Workload simulation on multiple extended EPC models in Bee-Up

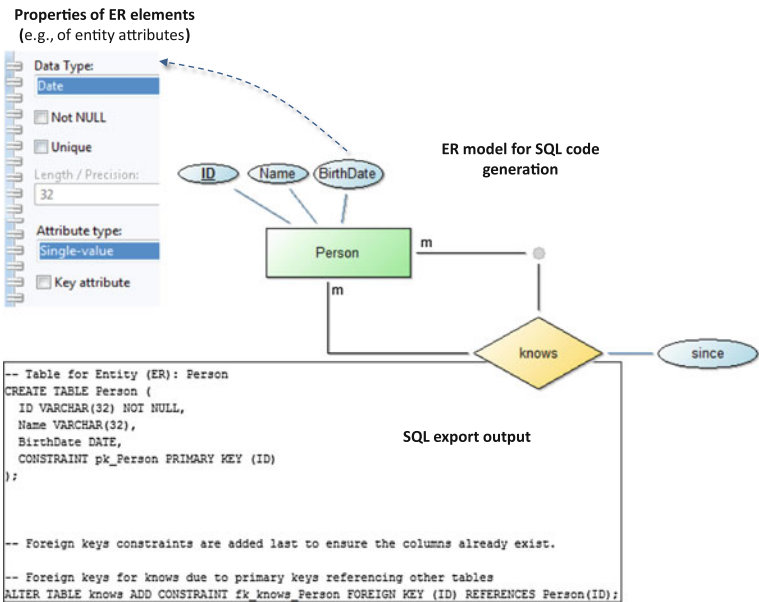


Fig. 14 ER modeling in Bee-Up, with SQL generation mechanism

6 Conclusions

The proposed FCML method addresses a heterogeneous domain by enabling multi-purpose modeling in the same tool, with varying degrees of domain-specificity added to several commonly used modeling languages that traditionally address the domains of software engineering and business process management. SQL generation, workload simulation and path analysis are typical examples of specific mechanisms that must be executed on particular model patterns and will provide useful results only if domain-specific properties are assigned to model elements, thus extending the scope of the languages incorporated in the FCML method. In this sense, its hybridization is not only a juxtaposition of types of diagrams from different languages, but it is also an integration of concepts with recurring semantics—at least for the purposes of process-based simulation. On-going work is being invested for an extensive semantic lifting across the languages included in FCML, since opportunities are open due to the different complementing scopes (e.g., the entity in an ER data model could be linked as input to an EPC function), but also due to certain overlapping (e.g., UML sequence diagrams could be linked as subprocesses to a higher level process model described with BPMN or EPC).

Ultimately, the kind of hybridization proposed by FCML and its proof-of-concept Bee-Up are aimed at being used as a multi-purpose and multi-layered modeling approach, where method agility is manifested by a multitude of notation alternatives in a single tool for different kinds of users, and also by machine-interpretable semantics on which functionality of varying specificity may be built.

The work at hand also advocates a possible starting point in the design of domain-specific modeling languages, while at the same time providing a resource of lessons learned which can support both teaching activities in the area of conceptual modeling, as well as scientific experimentation at metamodeling level with the fundamental modeling languages assimilated in FCML.

Acknowledgements We thank Srdjan Zivkovic and all the participants of the NEMO Summer School Series for the discussion of FCML.

Tool Download <http://www.omilab.org/bee-up>.

References

1. OMG: The BPMN specification page. <http://www.bpmn.org> (2016). Accessed 1 Mar 2016
2. Chen, P.: The entity-relationship model—toward a unified view of data. *ACM Trans. Database Syst.* **1**(1), 9–36 (1976)
3. Scheer, A.W.: ARIS, p. 20. Springer, Heidelberg, Vom Geschäftsprozess zum Anwendungssystem (2002)
4. Software AG: ARIS—the community page. <http://www.ariscommunity.com> (2016). Accessed 1 Mar 2016

5. OMG: The UML resource page. <http://www.uml.org> (2016). Accessed 1 Mar 2016
6. Petri, C.A., Reisig, W.: Petri net. *Scholarpedia* **3**(4), 6477 (2008). doi:[10.4249/scholarpedia.6477](https://doi.org/10.4249/scholarpedia.6477)
7. Reisig, W.: *Understanding Petri Nets*. Springer, Heidelberg (2013)
8. OMiLAB: The metamodeling page for FCML and the Bee-Up tool. <http://www.OMiLAB.org/bee-up> (2016). Accessed 1 Mar 2016
9. BOC GmbH: ADOxx—official website. <https://www.adoxx.org/live/home> (2016). Accessed 1 Mar 2016
10. Buchmann, R.A., Karagiannis, D.: Agile modelling method engineering: lessons learned in the ComVantage project. In: Ralyte, J., Espana, S., Pastor, O. (eds.) *Proceedings of the 8th IFIP WG 8.1 Conference on the Practice of Enterprise Modelling (PoEM 2015)*, Valencia, Spain. LNBP, vol. 235, pp. 356–373. Springer, Heidelberg (2015a)
11. Karagiannis, D.: Agile modeling method engineering. In: *Proceedings the 19th Panhellenic Conference on Informatics (PCI 2015)*, pp. 5–10, Athens, Greece. ACM (2015)
12. Krogstie, J., Sindre, G., Jorgensen, H.: Process models representing knowledge for action: a revised quality framework. *Eur. J. Inf. Syst.* **15**, 91–102 (2006)
13. Moody, D.: The physics of notations: towards a scientific basis for constructing visual notations in software engineering. *IEEE Trans. Software Eng.* **35**(5), 756–777 (2009)
14. Bencomo, N., France, R., Cheng, B.H.C., Abmann, U.: *Models@run.time*. LNCS, vol. 8378. Springer, Heidelberg (2014)
15. van der Aalst, W.M.P.: Process-aware information systems: lessons to be learned from process mining. In: Jensen, L., van der Aalst, W.M.P. (eds.) *Transactions on Petri Nets and Other Models of Concurrency II*. LNCS, vol. 5460, pp. 1–26. Springer, Heidelberg (2009)
16. Schmidt, D.C.: Model-driven engineering. *IEEE Comput.* **39**(2), 25–31 (2006)
17. Box, G.E.P.: *Science and Statistics*. J. Amer. Stat. Assoc. **71**, 791–799 (1976)
18. Ganter, B., Stumme, G., Wille, R. (eds.) *Formal Concept Analysis: Foundations and Applications*. LNAI vol. 3626, Springer (2005)
19. Staab, S., Studer, R.: *Handbook on Ontologies*. Springer (2004)
20. W3C: OWL 2—the W3C recommendation. <https://www.w3.org/TR/owl2-overview>. Accessed 1 Mar 2016
21. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F.: *Handbook of Description Logics*. Cambridge University Press (2010)
22. Staab, S., Walter, T., Gröner, G., Parreiras, F.S.: Model driven engineering with ontology technologies. In: Abmann, U., Bartho, A., Wende, C. (eds.) *Reasoning Web—Semantic Technologies for Software Engineering*, LNCS 6325, pp. 62–98. Springer, Heidelberg (2010)
23. Frank, U.: Multilevel modeling: toward a new paradigm of conceptual modeling and information systems design. *Bus. Inf. Syst. Eng.* **6**(6), 319–337 (2014)
24. Voultsidis, M.: ER2SQL—the official page. <http://www.er2sql.com> (2016). Accessed 1 Mar 2016
25. Andersson, M.: Extracting an entity-relationship schema from a relational database through reverse engineering. In: Loucopoulos, P. (ed.) *Proceedings of the 13th International Conference on the Entity-Relationship approach*, Manchester, England. LNCS, vol. 881, pp. 403–419. Springer, Heidelberg (1994)
26. Della, P.G., Di Marco, A., Intriglia, B., Melatti, I., Pierantonio, A.: Xere: towards a natural interoperability between XML and ER diagrams. In: Pezze, M. (ed.) *Proceedings of the 6th International Conference FASE 2003 part of the Joint European Conference on Theory and Practice of Software*, Warsaw, Poland. LNCS, vol. 2621, pp 356–371. Springer, Heidelberg (2003)
27. Liu, C., Li, J.: Designing quality XML Schemas from ER diagrams. In: Yu, J.X., Kitsuregawa, M., Leong, H.V. (eds.) *Proceedings of the 7th International Conference on Advances in Web-Age Information Management*, Hong Kong, China. LNCS 4016, pp 508–519. Springer, Heidelberg (2006)

28. Embley, D.W., Ling, T.W.: Synergistic database design with an extended Entity-Relationship model. In: Lochovsky, F.H. (ed.) *Proceedings of the 8th International Conference on Entity-Relationship approach to database design and querying*, pp. 111–128. Elsevier, Toronto, Canada (1990)
29. Teorey, T.J., Yang, D., Fry, J.P.: A logical design methodology for relational databases using the extended entity-relationship model. *ACM Comput. Surv.* **18**(2), 197–222 (1986)
30. Conceptual Modeling conference series. The ER conference series website <http://www.conceptualmodeling.org> (2016). Accessed 1 Mar 2016
31. Booch, G., Rumbaugh, J., Jacobson, I.: *Unified Modeling Language user guidelines*, 2nd edn. Addison-Wesley (2005)
32. OMG: The XMI specification page. <http://www.omg.org/spec/XMI> (2016). Accessed 1 Mar 2016
33. OMG: The OCL resource page. <http://www.omg.org/spec/OCL>. Accessed 1 Mar 2016
34. Carlson, D.: *Modeling XML Applications with UML*. Addison-Wesley (2001)
35. OMG: The SysML resource page. <http://www.omg.sysml.org> (2016). Accessed 1 Mar 2016
36. Vanderperren, Y., Mueller, W., He, D., Mischkalla, F., Dehaene, W.: Extending UML for electronic systems design: a code generation perspective. In: Nicolescu, G., O'Connor, I., Pigué, C. (eds.) *Design Technology for Heterogeneous Embedded Systems*, pp. 13–39. Springer, Netherlands (2012)
37. ACM/IEEE: Official page of the 18th edition of the MODELS International Conference. <http://cruise.eecs.uottawa.ca/models2015> (2015). Accessed 1 Mar 2016
38. OASIS: BPEL—the official website. https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel (2016). Accessed 1 Mar 2016
39. WfMC XPD L specification—official website (2015). <http://www.xpdl.org>. Accessed 1 Oct 2015
40. White, S.A.: Using BPMN to model a BPEL process. *BPTrends* **3**, 1–18 (2005)
41. Recker, J., Mendling, J.: On the translation between BPMN and BPEL: conceptual mismatch between process modeling languages. In: Latour, T., Petit, M. (eds.) *Proceedings of Workshops and Doctoral Consortium. The 18th International Conference on Advanced Information Systems Engineering*, pp. 521–532. Namur Univ. Press (2006)
42. zur Muehlen, M., Recker, J.: How much language is enough? Theoretical and practical use of the business process management notation. In: Bellahsene, Z., Leonard, M. (eds.) *Proceedings of the 20th International Conference on Advanced Information Systems Engineering*, Montpellier, France. LNCS vol. 5074, pp. 465–479. Springer, Heidelberg (2008)
43. OMG: The DMN specification page. <http://www.omg.org/spec/DMN> (2016). Accessed 1 Mar 2016
44. Velitchkov, I.: BPMN versus EPC revisited part 1. <http://www.ariscommunity.com/users/ivo/2011-04-11-bpmn-vs-epc-revisited-part-1> (2016). Accessed 1 Mar 2016
45. Burlton, R.: Perspectives on Process Modeling. *BPTrends* (2009).
46. Swenson, K.: BPMN 2.0: no longer for business professionals. <https://social-biz.org/2010/09/01/bpmn-2-0-no-longer-for-business-professionals/> (2016). Accessed 1 Mar 2016
47. zur Aalst, W.M.P.: Formalization and verification of event-driven process chains. *Inf. Softw. Technol.* **41**(10), 639–650 (1999)
48. Meertens, L.O., Iacob, M.E., Eckartz, S.M.: Feasibility of EPC to BPEL model transformations based on ontology and patterns. In: Rinderle-Ma, S., Sadiq, S., Leymann, F. (eds.) *Proceedings of the BPM 2009 workshops*, Ulm, Germany. LNBIP, vol. 43, pp. 347–358. Springer, Heidelberg (2010)
49. Mendling, J., Nüttgens, M.: EPC markup language: an XML-based interchange format for event-driven process chains. *IseB* **4**(3), 245–265 (2006)
50. Störrle, H.: Semantics of control-flow in UML 2.0 activities. In: Bottoni, P., Hundhausen, C., Levialdi, S., Tortora, G. (eds.) *Proceedings of the 2004 IEEE Symposium on Visual Languages and Human-Centric Computing*, pp. 235–242. IEEE, Rome, Italy (2004)

51. Jensen, K., Kristensen, L.M.: Coloured Petri nets. Springer, Heidelberg (2009)
52. Dufourd, C., Finkel, A., Schnoebelen, P.: Reset nets between decidability and undecidability. In: Larsen, K.G., Skyum, S., Winskel, G. (eds.) Proceedings of the 25th Int Colloquium ICALP98, Aalborg, Denmark. LNCS, vol. 1443, pp. 103–115. Springer, Heidelberg (1998)
53. Eclipse: The Eclipse Modelling Framework official page. <https://eclipse.org/modeling/emf/> (2016). Accessed 1 Mar 2016
54. Jeusfeld, M.: Metamodeling and method engineering with ConceptBase. In: Jeusfeld, M., Jarke, M., Mylopoulos, J. (eds.) Metamodeling for Method Engineering, pp. 89–168. The MIT Press, Cambridge, USA (2009)
55. Kelly, S., Lyytinen, K., Rossi, M.: MetaEdit + a fully configurable multi-user and multi-tool CASE and CAME environment. In: Bubenko, J., Krogstie, J., Pastor, O., Pernici, B., Rolland, C., Solvberg, A. (eds.) Seminal Contributions to Information Systems Engineering, pp. 109–129. Springer
56. MetaCase: MetaEdit + tool. <http://www.metacase.com/products.html> (2016). Accessed 1 Mar 2016
57. W3C: The RDF official resource page. <http://www.w3.org/RDF/> (2016). Accessed 1 Mar 2016
58. W3C: The RIF specification page. <https://www.w3.org/TR/rif-overview/> (2016). Accessed 1 Mar 2016
59. Visic, N., Fill, H.-G., Buchmann, R., Karagiannis, D.: A domain-specific language for modelling method definition: from requirements to grammar. In: Rolland, C., Anagnostopoulos, D., Loucopoulos, P., Gonzalez-Perez, C. (eds.) Proceedings of the 9th International Conference on Research Challenges in Information Science (RCIS 2015), pp. 286–297. IEEE, Athens, Greece (2015)
60. OMG: The MOF specification page. <http://www.omg.org/mof/> (2016). Accessed 1 Mar 2016
61. Kelly, S., Tolvanen, J.P.: Domain-Specific Modeling: Enabling Full Code Generation. Wiley (2008)
62. Budinsky, F., Steinberg, D., Merks, E., Ellersick, R., Grose, T.J.: Eclipse Modeling Framework. Addison Wesley, The Eclipse Series (2004)
63. Kern, H., Hummel, A., Kuhne, S.: Towards a comparative analysis of meta-metamodels. In: The 11th Workshop on Domain-Specific Modeling, Portland, USA (2011). <http://www.dsmforum.org/events/DSM11/Papers/kern.pdf>. Accessed 1 Oct 2015
64. Zivkovic, S.: Metamodel composition in hybrid modelling—a modular approach. Doctoral thesis, University of Vienna (2016)
65. Karagiannis, D., Kühn, H.: Metamodelling platforms. In: Bauknecht, K., Min Tjoa, A., Quirmayer, G (eds.) Proceedings of the Third International Conference EC-Web 2002—DEXA 2002, Aix-en-Provence, France. LNCS vol. 2455, p. 182. Springer (2002)
66. Karagiannis, D., Buchmann, R.A.: Model fragment comparison using natural language processing techniques. In: Hess, T. (ed.) Brenner W, pp. 249–269. Wirtschaftsinformatik in Wissenschaft und Praxis, Springer (2014)
67. Harel, D., Rumpe, B.: Modeling Languages: Syntax, Semantics and All That Stuff, Part 1: The Basic Stuff (2000)
68. Engels, G., Hausmann, J.H., Heckel, R., Sauer, S.: Dynamic meta modeling: a graphical approach to the operational semantics of behavioral diagrams in UML. In: «UML» 2000—The Unified Modeling Language, pp. 323–337. Springer, Berlin Heidelberg (2000)
69. Walter, T., Parreiras, F.S., Staab, S.: OntoDSL: an ontology-based framework for domain-specific languages. In: Schürr, A., Selic, B. (eds.) Proceedings of the 12th International Conference on MODELS, Denver, USA. LNCS vol. 5795, pp. 408–422. Springer, Heidelberg (2009)
70. Buchmann, R.A., Karagiannis, D.: Modelling mobile app requirements for semantic traceability. J. Requirements Eng. (2015). doi:[10.1007/s00766-015-0235-1](https://doi.org/10.1007/s00766-015-0235-1)

71. Karagiannis, D., Buchmann, A.: Linked open models: extending linked open data with conceptual model information. *Inf. Syst.* **56**, 174–197 (2016)
72. Fill, H.G., Karagiannis, D.: On the conceptualisation of modelling methods using the ADOxx meta modelling platform. *Enterp. Model. Inf. Syst. Architect.* **8**(1), 4–25 (2013)

Domain-Specific Conceptual Modeling

Concepts, Methods and Tools

Karagiannis, D.; Mayr, H.C.; Mylopoulos, J. (Eds.)

2016, XII, 594 p. 301 illus., Hardcover

ISBN: 978-3-319-39416-9