Example-Driven Modeling Using Clafer

Michał Antkiewicz, Kacper Bąk, Krzysztof Czarnecki, Zinovy Diskin, Dina Zayan[1], and Andrzej Wąsowski[2]

[1] GSD Lab, University of Waterloo, Canada
{mantkiew,kbak,kczarnec,zdiskin,dzayan}@gsd.uwaterloo.ca
[2] IT University of Copenhagen, Denmark
wasowski@itu.dk

Abstract. Example-driven modeling (EDM) is an approach to systematically using explicit examples for eliciting, modeling, verifying, and validating complex business knowledge. In EDM, examples and abstractions are equally important parts of the model, as both are needed for effective knowledge transfer (*model = examples + abstractions*). We show how Clafer, a lightweight structural modeling language, can be used when applying EDM for domain analysis and requirements elicitation. We present a sample modeling scenario and features of Clafer which support EDM.

1 Introduction

Building upon results from cognitive psychology and software engineering, we have proposed example-driven modeling (EDM) [7], a modeling approach which prescribes using explicit examples for eliciting, modeling, verifying & validating domain knowledge. Despite examples being widely used in behavioral modeling, end-user programming, grammar and schema inference and testing, software specification and testing, their use in structural modeling is much less common.

We have proposed two hypotheses related to **why** examples should be used (H1 & H2) and four hypotheses related to **how** they should be used (H3-H6):

H1 *Constructing models with the aid of explicit examples improves the quality of models.*
H2 *Augmenting models with explicit examples improves model comprehension among various stakeholders.*
H3 *EDM starts either with abstractions or examples. A modeler typically goes back and forth between the two.*
H4 *If examples express requirements for using abstractions, then automated tools validate models on examples to discover errors.*
H5 *A variety of generated examples leads to more effective model construction, comprehension, and validation.*
H6 *Using both positive and negative examples leads to more precise models than positive examples alone.*

We invite the community to participate in the effort of validating these hypotheses as more experience and economic data as well as guidance for applying

EDM are still needed. We have performed a controlled experiment to validate the hypothesis **H2** and partially **H6** [17].

In this paper, we assume that these hypotheses are true and we show how EDM can be performed using a lightweight structural modeling language Clafer [1, 5] and a dedicated reasoner, Clafer Instance Generator (ClaferIG) [1]. We present a domain knowledge elicitation scenario and introduce Clafer and its features supporting EDM in Section 2. We identify *redefinition* [3, 6] as a key semantic mechanism which enables the unification of examples and abstractions in Clafer's syntax. We then discuss the relationship between Clafer and the hypotheses of EDM in Section 3. EDM needs three key parts: methods, languages, and tools. We provide Clafer as a language designed to support EDM and ClaferIG as a supporting tool for example derivation. These two enable us to gather experience and data needed to validate the hypotheses and define EDM methods.

2 A Domain Knowledge Elicitation Scenario

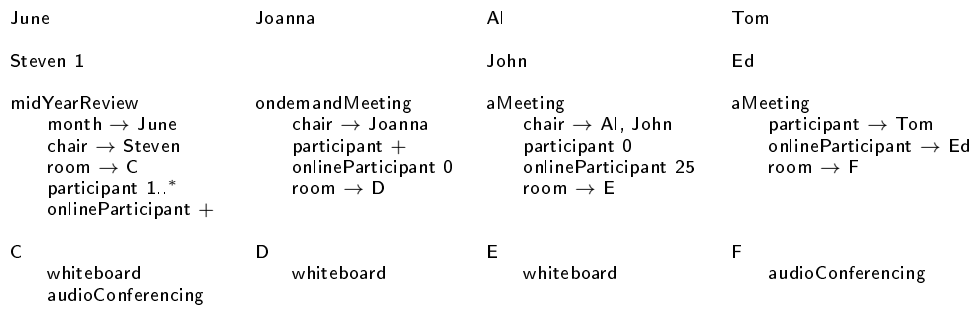
In this section, we describe a domain knowledge elicitation scenario whereby a business analyst (BA), Bob, elicits domain knowledge from a subject-matter expert (SME), Alice. The scenario is based on the one we previously presented [7]; however, here we elaborate on the scenario and demonstrate how it can be performed using Clafer. Alice's organization needs a room booking system.

ALICE: Our members often book rooms for meetings. We need to manage the available rooms and to provide room-booking functionality.

BOB: Give me an example of a room booking, please.

ALICE: We have a mid-year review meeting in June. It is organized by Steven, a chair, and is held in the meeting room C that provides a whiteboard and audioconferencing equipment to include online participants.
[BOB *models the example using Clafer (Fig. 1(a) explained below).*]

In Clafer, the model is built from only one kind of element, called *clafer*, which can play multiple roles, such as, representing a type, an attribute, a relationship, an instance, or a combination thereof. Each clafer has a name and it can be either top-level or nested under other clafers. Just writing a name in a new line, such as `June`, declares a new clafer (a top-level one in this case). Clafer nesting is expressed via indentation, e.g., `whiteboard` is nested under `C`. Each clafer has a number of possible occurrences called *clafer cardinality*, such as, 0, 0..1 (? for short), 1, 0..$*$ ($*$ for short), 1..$*$ (+ for short), and $n..m$ (e.g., 2..4). By default, the cardinality is 1. For example, `June` has cardinality 1 by default, whereas `Steven` has cardinality 1 explicitly specified. Also, both `participant` and `onlineParticipant` have clafer cardinality 1..$*$; however, for the latter the cardinality is specified using the shorthand notation +. Finally, clafers can refer to other clafers using $\rightarrow$. For example, the clafer `room` specifies that one can navigate from `midYearReview` to `C` via `room`.



(a) Mid-year review (b) On-demand (c) Negative 1 (d) Negative 2

Fig. 1. Concrete Examples of Meetings

ALICE: Another example is an on-demand meeting organized within work hours. Joanna, a team-leader, sometimes meets other team members in room D. They use a whiteboard and have no online participants.
[BOB *notes the example (Fig. 1(b)). The number of* `onlineParticipants` *is 0.*]
[BOB *observes some similarities between the two examples and creates two examples to clarify a few details with* ALICE.]

BOB: Are examples (c) and (d) in Fig. 1 valid?

ALICE: No, both examples are invalid. In example (c), there are two chairs, but a meeting must have exactly one chair. Also, it is our policy that rooms cannot be booked for meetings with online participants only. The maximum number of online participants supported by our system is 20. Finally, one cannot have a meeting with online participants without audio conferencing equipment! In example (d), there's no chair, which is required. I am also unsure if we have any rooms without a whiteboard.

BOB: Thank you, that clarifies a lot of details! I think we are now ready to create a more abstract model.
[BOB *creates a few abstractions (Fig. 2(a)).*]

BOB: We have three main abstractions: `Member`, `Room`, and `Meeting`. We can now extend the examples from Fig. 1(a)(b) by adding types. In Fig. 2(a), we can see that `Steven` is a `Member` and that `participant` points to a `Member`.

In Clafer, abstractions are created using a keyword `abstract`. An abstract clafer is not an example — it aggregates commonalities of a set of examples. Clafers defined without keyword `abstract` are called *concrete clafers*. Clafers can inherit from at most one clafer using `:`, e.g., `C : Room` [3]. Clafers can also use other clafers to specify the type of a reference using →, e.g., `participant → Member`. Both

[3] Clafer does **not** unify inheritance with instantiation: `:` is only used for inheritance. Clafer does not represent instances directly; instead, in the semantics of Clafer, concrete clafers correspond to partial instances encoded as (often singleton) classes [3, 6].

```
abstract Member

abstract Room

abstract Meeting

June
Steven : Member
midYearReview : Meeting
    month → June
    chair → Steven
    room → C
    participant → Member +
    onlineParticipant → Member 1..20
C : Room
    whiteboard
    audioConferencing

Joanna : Member
ondemandMeeting : Meeting
    chair → Joanna
    room → D
    participant → Member +
    onlineParticipant → Member 0
D : Room
    whiteboard
```

(a) Adding partial types

```
abstract Member

abstract Room
    whiteboard ?
    audioConferencing ?

abstract Meeting
    chair → Member
    room → Room
    participant → Member +
    onlineParticipant → Member 0..20
    [ some onlineParticipant ⟹ some room.audioConferencing ]

June
Steven : Member
midYearReview : Meeting
    month → June
    chair → Steven
    room → C
    onlineParticipant → Member 1..20
C : Room
    whiteboard
    audioConferencing

Joanna : Member
ondemandMeeting : Meeting
    chair → Member = Joanna
    room → D
    onlineParticipant → Member 0
D : Room
    whiteboard
```

(b) Refining the abstractions. Examples by redefinition

Fig. 2. Adding abstractions

concrete and abstract clafers can be inherited from (:) and pointed to (→). Adding types as in Fig. 2(a) is useful for classifying the examples. For example, it was not clear what the clafer `C` represented; now it is clear that it is a `Room`.

BOB: From our four examples and the discussion, we can now see that every meeting must have exactly one chair who is a member; every meeting is held in one room; there must be at least one or more participants; and there may be up to 20 online participants - we can move them into the abstraction. For now, I feel that `month` is something specific to `midYearReview`, so let's leave it there.
Regarding rooms, it is clear that `audioConferencing` has cardinality 0..1. However, Alice was uncertain whether all rooms have a `whiteboard`, so until it is verified we assume cardinality 0..1 as well.
[BOB *factors out the common parts into the abstractions (Fig. 2(b)). He also added a constraint to* `Meeting` *that the room used for a given meeting must have* `audioConferencing` *equipment if the meeting has* `onlineParticipants`.]

The examples in Fig. 2(b) are constructed using *redefinition* [3, 6] of the clafers inherited from the abstractions, which allows them to look exactly as the original examples from Fig. 2(a). Using redefinition a modeler can restrict any aspect

of the inherited clafer's declaration. Here, we show two ways of restricting the inherited references: 1) by redefinition and restricting the type, as in `chair` → `Steven`, whereby the type is restricted from `Member` to `Steven`, and 2) by constraining to a particular set using `=`, as in `chair` → `Member = Joanna`, whereby `chair` can still point to any `Member` but it is constrained to only point to `Joanna`. Also, `onlineParticipant -> Member 0` illustrates restricting clafer cardinality. Finally, since the examples do not redefine the inherited clafer `participant`, it is omitted (cf. Fig. 2a).

Clafer constraints are specified between [and] and can be arbitrary first-order predicate logic formulas with some relational logic operators, e.g., *join* (`.`). Clafer's constraint language is heavily inspired by the constraint language of Alloy [9]; it does not distinguish between scalar values and sets—everything is a set. In our example, the constraint can be read as follows: "*having some `onlineParticipants` implies `audioConferencing` equipment in the `room`.*"

All examples provided by ALICE and BOB are *partial examples*, i.e., they provide full detail about, e.g., `month` and `room`, but they do not provide details about the on-site and on-line participants, they only indicate their number. Partial examples can be *completed* under closed-world assumption (CWA) or open-world assumption (OWA). In Clafer, we apply the CWA interpretation in reasoning, where the reasoner only fills in the missing information. For example, the room `D` only lists a `whiteboard` and omits `audioConferencing` equipment, which is interpreted as underspecified, that is, the room may or may not have the equipment. To specify that the equipment is not present, BOB would have to write `audioConferencing 0`. Modelers, on the other hand, naturally apply the OWA interpretation and freely extend the examples as shown in Fig. 2a. Therefore, adding abstractions is helpful for indicating what details can be specified, partially specified, or omitted in examples and what details can be completed by the reasoner under CWA.

BOB: So far, our abstractions look quite good. Let's automatically derive a few examples to validate our model. Alice, are they valid?

[BOB *creates an underspecified meeting,* `aMeeting`*, which has some on-line participants (Fig. 3(a)), together with* `aRoom` *and between 2 to 5* `aMembers`*. He uses Clafer Instance Generator (ClaferIG)[1] to automatically generate all possible completions of* `aMeeting`*,* `aRoom`*, and* `aMember` *under CWA. Two completions are shown in Fig. 3(b)(c)*]

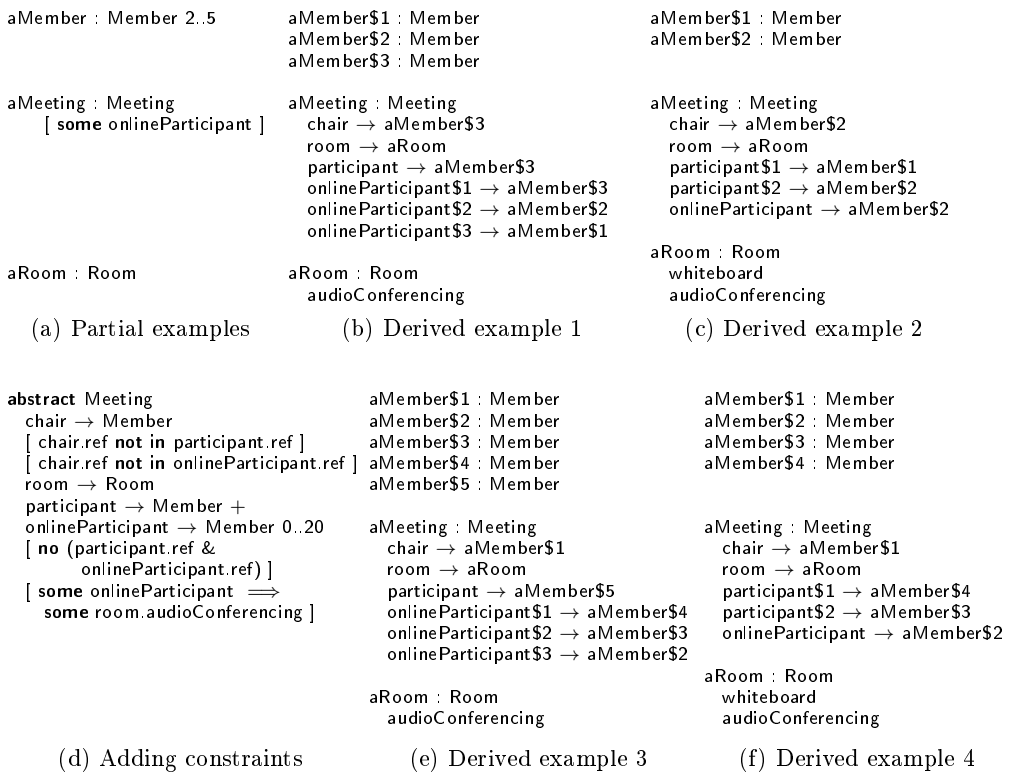
ALICE: No, the examples (b) and (c) are invalid! A chair cannot be a participant. Also, no member can participate both on-site and on-line.

BOB: Then we need to add the missing constraints.

[BOB *adds the following three constraints: 1) the chair cannot be a participant, 2) the chair cannot be an on-line participant, and 3) the sets of participants and on-line participants are disjoint (Fig. 3(d)).* BOB *derives examples Fig. 3(b)(c)*]

ALICE: Examples (b) and (c) are valid.

[*Finally,* BOB *uses ClaferIG to verify the abstractions against the negative examples from Figure 1(c)(d) (after adding types) and from Figure 3(b)(c)—the tool reports constraint violations confirming that the examples are negative.*]



(a) Partial examples (b) Derived example 1 (c) Derived example 2

(d) Adding constraints (e) Derived example 3 (f) Derived example 4

Fig. 3. Automatic example derivation

3 Discussion: Clafer & The Hypotheses of EDM

Here, we discuss the relationship between how Clafer was used in the presented scenario and the hypotheses of EDM.

H1 *Constructing models with the aid of explicit examples improves the quality of models.* We illustrated that the examples helped to ground the discussion, explore corner cases, derive initial abstractions and refine them by discovering the missing constraints, discover points of uncertainty, and verify whether the abstractions allow positive examples and disallow the negative ones. Also, the examples provided by the SME were often partial and Clafer allowed the modeler (here, the BA) to express them naturally (nothing special required).

H2 *Augmenting models with explicit examples improves model comprehension among various stakeholders.* The modeler wrote both the abstractions and examples in Clafer as a part of the same model. In general, SMEs cannot be expected to be proficient in understanding the abstractions, especially when complex constraints are involved or a notation for abstractions differs significantly from the notation for examples. The modelers, on the other hand, can work directly with

abstractions and only use examples to explore corner or exceptional cases. Therefore, when the examples are part of the model, they provide a shared basis for communication between expert modelers and other stakeholders.

H3 *EDM starts either with abstractions or examples. A modeler typically goes back and forth between the two.* In EDM, the modelers perform two main activities that relate examples with abstractions: 1) *abstraction inference* (AI) – for synthesizing abstractions from a set of examples, 2) *example derivation* (ED) – for generating examples from abstractions. During AI, modelers compare examples, observe commonalities and differences, use negative examples to illustrate constraint violations, determine overconstraining and underconstraining of abstractions. During ED, modelers create concrete or partial positive and negative examples, automatically derive completions of partial examples, and validate the examples with the SMEs. In Clafer, both starting with abstractions or starting with examples are possible. ClaferIG supports deriving concrete examples from abstractions and partial examples. The unified syntax for abstractions and examples of Clafer allows the modeler to easily compare examples and pull up common content of examples to the abstractions by simply copying and pasting. At the same time, the modeler can leave the examples mostly unchanged, except for providing type information, because of redefinition. Thus, in Clafer evolving examples into abstractions requires minimal effort.

H4 *If examples express requirements for using abstractions, then automated tools validate models on examples to discover errors.* Using ClaferIG, a modeler can check consistency of the abstractions and verify whether the positive examples are allowed and the negative examples are disallowed. In the case of an inconsistency, ClaferIG helps the modeler in diagnosing them by reporting a near-miss example; this capability is realized by computing UnSAT cores and removing one or more contradicting constraints from each core [10].

H5 *A variety of generated examples leads to more effective model construction, comprehension, and validation.* We illustrated how partial specialization allows deriving a variety of examples, while giving the user control over the kinds of examples that are generated.Automatic derivation of examples exposes hidden assumptions and constraints, as the tool can exhaustively explore all cases.

H6 *Using both positive and negative examples leads to more precise models than positive examples alone.* We illustrated in the scenario that using the negative examples allows the modeler to discover constraints because the SME can catch constraint violations when inspecting the examples. Furthermore, Bob validated the model by checking that the negative examples indeed violate the constraints.

4 Related Work

We discussed the related work on EDM in [7]. Here we only focus on language and tool support for EDM.

We have performed a controlled experiment to evalue the effects of using examples on structural model comprehension [17]. This experiment confirmed the hypothesis **H2** and showed that using explicit examples has significant pos-

itive effect on the effectiveness of the experiment participants in comprehending a structural domain model, their ability to correctly, completely, and efficiently instantiate the model, and correctly and efficiently answer a number of questions. Furthermore, both positive and negative partial examples designed to illustrate a specific point were very helpful for the participants in their ability to complete the experimental tasks (cf. **H6**).

Alloy [9] is a lightweight modeling language supported by a reasoner. Alloy models specify abstractions; the reasoner derives examples and checks model consistency. Although Alloy can express partial examples by encoding them as singleton sets, a dedicated syntactic extension has been proposed [13]. In contrast with Clafer, Alloy provides no first-class support for redefinition and for encoding hierarchical models (in Clafer, clafers can be nested arbitrarily deeply); however, both can be realized indirectly by specifying model constraints.

UML [14] class and object diagrams are two standardized notations for expressing abstractions and examples, respectively. They are two separate diagrams that exist independently (separate meta-classes and cross-referencing in certain ways is restricted, for example, an object cannot be used as a type of a reference and links cannot be made between objects and classes). In practice, relating UML object to class diagrams is troublesome, because it requires the modeler to constantly switch attention between separate diagrams. In Clafer, abstractions can be mixed and cross-referenced with examples and, thanks to syntactic unification, examples can be easily evolved into abstractions by copy/paste.

UML class diagrams is a syntactically much richer notation than Clafer. While Clafer reduces structural modeling concepts to *clafer*, UML class diagrams make each concept explicit both in the syntax and semantics (e.g., different types of associations). Both languages offer support for redefinition and for partial examples. UML object diagrams offer a limited support for partial examples. Whereas they can express uncertainty of slots' values, they cannot express uncertainty about existence of objects. Clafer can model both. In [3], we formally define and discuss partial instances, the notions of redefinition and instance extension in the CWA and OWA frameworks. Consequently, we show how partial instances can be encoded in UML via singleton classes and subclassing.

UML object diagrams can express positive examples. Modal object diagrams [11] is an extension that also adds negative examples. Relating negative examples to abstractions is difficult unless the modeler knows why they are inconsistent with abstractions. We are not aware of any non-academic UML tools that can verify consistency of object and class diagrams. ClaferIG can verify the models and can additionally point to inconsistent model elements; however, currently Clafer does not support expressing the *negative* modality.

We originally introduced Clafer in [5]. Our recent work [6] is a substantial revision of the language. In formal semantics, we explained the great flexibility of the concept *clafer* and clarified some language constructs (e.g., references). Furthermore, we formally explained the notion of redefinition among clafers, which can be realized at any nesting level in the containment hierarchy and which allows for encoding partial examples as subclasses of abstractions. However, in [6]

we do not show how redefinition can be applied to unify the syntax of examples and abstractions to support EDM, which is shown in this paper.

Clafer has been originally designed as a language for modeling variability in product lines (PLs). In [2], we demonstrate how to use Clafer to migrate from a clone-and-own software family to a product line. The presented approach is example-driven and involves abstraction inference from a set of examples, each representing an individual product. Furthermore, in the context of PLs examples are also useful for validation of the PL variability models. While the previous work [2] has focused on simple Boolean feature models, this work addresses EDM for a broader application scope and much richer structural models.

There is a vast number of works related to inferring abstractions from examples that could be leveraged for Clafer. For example, inferring grammars [15], metamodels [8], data schemas [4], behavioral [16] and structural models [12].

5 Conclusion

We presented how Clafer and ClaferIG can be used in an EDM modeling scenario and how Clafer relates to the EDM hypotheses. Although usually only experts can specify abstractions directly, both experts and novices can understand examples well. The use of abstractions and examples differs for experts and novices. The former naturally start with abstractions and use examples to clarify difficult details. Novices, on the other hand, prefer to start with examples and only then infer abstractions that generalize examples. It is clear that a specific variety of examples is needed for an effective knowledge transfer. Most notably, positive examples demonstrate correct scenarios; near-miss negative examples show constraint violations, focusing on one case at a time; and partial examples allow focusing on a specific knowledge area.

In our view Clafer is well-suited for EDM due to 1) being based on a single concept, *clafer*, that can simplify co-evolution of abstractions and examples, 2) uniform syntax for encoding examples and abstractions so that both can be easily mixed in the same model, which is enabled by redefinition (such mixing is not supported by mainstream languages, such as UML class and object diagrams, and XML documents and XSD schemas), 3) uniform syntax which also allows for both examples and abstractions to look structurally similar, making evolution of examples into abstractions trivial, 4) support for partial examples that naturally express stakeholders partial view of the world, 5) support for specializing and extending both the examples and abstractions (easy extension and specialization by redefinition, constraints, and partial typing), 6) tool support that makes the model playable: constraints checking and automatic derivation of completions.

Future Work Despite the usefulness of examples, it is still unclear *what type of examples*, *how to collect them*, *how many are needed*, and *how diverse examples should be* to best support model comprehension and validation. Together with the presented hypotheses, empirical studies are needed to evaluate the effectiveness of EDM and to provide concrete guidelines for modelers.

Clafer design includes work on formal semantics and the development of the compiler. Some results from the formal semantics, e.g., redefinition, are currently partially implemented. We are also planning to implement the ability to mark top-level clafers as *negative* and extend the reasoner to check assertions that such instances are expected to violate constraints. Currently, Clafer supports only structural modeling; behavioral modeling in Clafer is an ongoing effort.

EDM requires a broad range of tools. Although tools can already automatically derive a huge number of examples, they provide little support for exploration and visualization of a set of examples (e.g., showing commonalities and differences, filtering by some characteristic). We have implemented a tool for simultaneous exploration of a set of examples, called Clafer Configurator [1]; however, it is currently restricted to the *attributed feature models with inheritance* subset of Clafer and it is not clear how it could be extended to support full Clafer. Abstraction inference is a challenging task, because abstractions must be inferred from an incomplete set of examples. Finally, since both abstractions and examples constitute a model, the tools should support their co-evolution.

References

1. Clafer Homepage, http://clafer.org
2. Antkiewicz, M., Bąk, K., Murashkin, A., Olaechea, R., Liang, J., Czarnecki, K.: Clafer tools for product line engineering. In: Software Product Line Conf. (2013)
3. Bąk, K., Diskin, Z., Antkiewicz, M., Czarnecki, K., Wąsowski, A.: Partial instances via subclassing. In: Intl. Conference on Software Language Engineering (2013)
4. Bex, G.J., Neven, F., Vansummeren, S.: Inferring XML schema definitions from XML data. In: International Conference on Very Large Data Bases (2007)
5. Bąk, K., Czarnecki, K., Wąsowski, A.: Feature and meta-models in Clafer: mixed, specialized, and coupled. In: Software Language Engineering (2010)
6. Bąk, K., Diskin, Z., Antkiewicz, M., Czarnecki, K., Wąsowski, A.: Clafer: Unifying class and feature modeling. Journal paper. Submitted for review. (2013)
7. Bąk, K., Zayan, D., Czarnecki, K., Antkiewicz, M., Diskin, Z., Wąsowski, A., Rayside, D.: Example-driven modeling. Model = Abstractions + Examples. In: New Ideas and Emerging Results (NIER) track of ICSE'13 (2013)
8. Cho, H.: A Demonstration-Based Approach for Domain-Specific Language Creation. Ph.D. thesis, University of Alabama (2013)
9. Daniel, J.: Software Abstractions: Logic, Language, and Analysis. MIT Press (2011)
10. Liang, J.: Correcting Clafer models with automated analysis. Tech. Rep. GSDLab-TR 2012-04-30, University of Waterloo (04/2012 2012)
11. Maoz, S., Ringert, J., Rumpe, B.: Modal object diagrams. In: ECOOP (2011)
12. Mendel, L.: Modeling by example. Master's thesis
13. Montaghami, V., Rayside, D.: Extending Alloy with partial instances. In: ABZ'12
14. OMG: OMG Unified Modeling Language (2011)
15. Parekh, R., Honavar, V.: Grammar inference, automata induction, and language acquisition. In: Handbook of Natural Language Processing (2000)
16. Piterman, N., Pnueli, A., Sa'ar, Y.: Synthesis of reactive (1) designs. In: Verification, Model Checking, and Abstract Interpretation (2006)
17. Zayan, D., Antkiewicz, M., Czarnecki, K.: Effects of using examples on structural model comprehension: A controlled experiment (2014), submitted for review.