OpenCASE– A Tool for Ontology-Centred Conceptual Modelling

Robert Pergl and Jakub Tůma

Department of Information Engineering,
Faculty of Economics and Management,
Czech University of Life Sciences, Prague, Czech Republic
{pergl,jtuma}@pef.czu.cz

Abstract. OpenCASE, an original CASE tool supporting conceptual modelling is presented in this paper. The CASE tool has been developed during the research focused on the ontology-centred conceptual modelling. It provides a strong emphasis on terms and their relations while supporting standard notations (now BORM, other notations are planned). The tool has an open plug-in-based architecture founded on the Eclipse platform, which makes the tool modular and extensible. The knowledge base of the models may be accessed via an API and thus used to implement verifications, various calculations (statistics), to transform models to outputs (reports) and to make inner transformations (e.g. normalisation). The architecture of the tool is briefly mentioned as well.

Keywords: CASE Tool, Eclipse platform, conceptual modelling, ontological analysis, BORM method.

1 Introduction

This contribution addresses the discussion of the importance of diligent ontological analysis during the enterprise IS modelling presented in [13], where the author explains the importance of ensuring the *consistency* between various models (and inside each model) and concludes (besides others) the need of a "quality CASE tool support".

In this paper, we would like to present our advancements in designing and implementing a CASE tool to support ontology-centred modelling: OpenCASE. OpenCASE [12] is a CASE tool designed to support the research in the field of conceptual modelling and ontologies. It is built upon the Eclipse framework [8] and it utilizes many of its advanced possibilities (see section 4). Right now, we have implemented the BORM method's Business Architecture Diagrams and Object Relation Diagrams ([7],[1], [10]) as a proof-of-concept of ontology-centred modelling and OpenCASE's philosophy and design[1].

[1] Apologies for the readers: due to lack of space in this short paper, we do not provide a BORM introduction here.

M. Bajec and J. Eder (Eds.): CAiSE 2012 Workshops, LNBIP 112, pp. 511–518, 2012.

2 Goal and Methodology

The goal of the contribution is to present `OpenCASE`, an original ontology-centred CASE tool. We present the philosophy behind the tool, its practical utilisation and its architecture.

3 Ontology-Centred Modelling

3.1 Entities vs. Elements

To deal with ontology generally means to deal with *terms* and their *relations*[2]. Conceptual modelling using the ontology-centred approach thus needs to fully support *tracking of distinct terms and their relations* throughout the models. This practically means a transition from *visually-centred* to the *ontology-centred* CASE tools architecture. We may find notions of this approach in some CASE and Meta-CASE tool like Craft.CASE [3], MetaEdit+ [11] and others, however we were focused on total concept purity in separating the Domain Layer and the Domain Model Layer while maintaining the relation between the elements (Figure 1[3]).

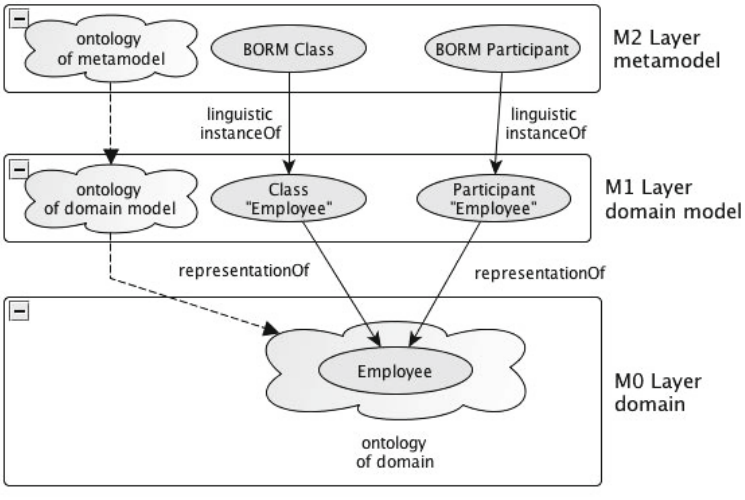

Fig. 1. Layered architecture of ontologies (taken from [13])

We implemented the concept of *ontologically equivalent elements* introduced in[13]:

[2] A more thorough introduction to ontologies and their relation to conceptual modelling and the BORM methodology may be found in our original paper [13].

[3] The layered architecture of ontologies has already been published in its rudimentary form numerus times over the past 30 years, see e.g. [4].

Definition 1. *We say that the model element x is* ***ontologically equivalent*** *to the model element y if and only if there exist relations $representationOf(x,t)$ and $representationOf(y,t)$, where t is an element of the CMoD*[4].

We implemented this concept by strictly discerning the domain *entities* and model's *elements*. An "entity" represents a domain object in the M0 layer in Figure 1, while the "element" is an object in the M1 layer, being a particular instance of the M2 layer element. Practically, let us suppose we deal with a `Customer` entity being modelled by a Participant `Customer` graphically represented according to the BORM methodology as a rectangle with a solid border and pale blue filling, which is an instance of the `Participant` concept.

There is a relation 1:N between entities and elements: each entity (layer M0) may be represented by several elements (layer M1): a Customer may play its role in several diagrams, thus being represented by a visual element in each, while being the same entity's *representationOf*. There is a screenshot in Figure 2 showing the `OpenCASE`'s support for tracking entities with respect to elements. The left panel shows all the entities. If we unfold an entity, we see all its elements. In Figure 2 there are 3 participant elements that represent entity `Customer`[5]. When we click an element, its full path is revealed in the status line. Double-clicking an element takes us to the appropriate diagram and selects the element.

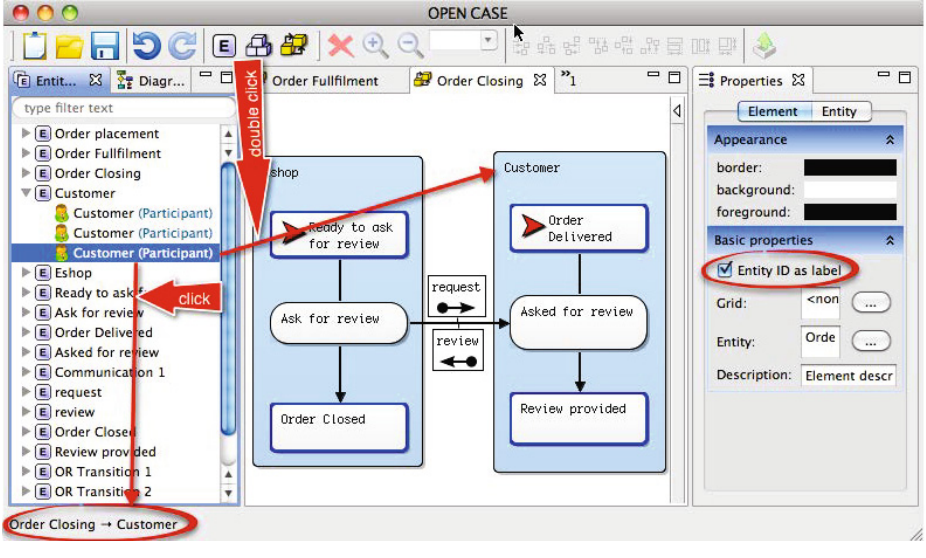

Fig. 2. Entities and Elements in `OpenCASE`

[4] CMoD = Concept Map of Domain ... a graph of domain terms and their relations.

[5] Generally, there may be various element types representing one entity, e.g. there may be as well a data class `Customer` that would describe customer's attributes.

Tracking the entities is a concept that enables us to:

Ensure elements' consistency – Elements usually take the name of their entity, so renaming an entity automatically renames all the attached elements. Thus if we have an element `Customer` participating in several diagrams and we change its entity's name to `Client`, all the attached elements get automatically renamed. In case we do not want an element to automatically take its entity's name – this may be the situation in multilingual diagrams or languages that use inflection – we may disable the implication "entity name → element name" (option `Entity ID as label`, red ellipse on the right in Figure 2). We need to rename the element by hand, however with the comfort of having the list of all elements that are *representationOf* the entity.

Facilitate impact analysis – When some change occurs to a domain entity, we may easily track the impact to the model, i.e. the elements in the model that may need attention due to the change.

Use the model knowledge base – Elements represent some information we have about the entity – its roles and relations in the domain. We may design and run reports, statistics, optimizations and reasoning based on this knowledge. `OpenCASE` provides a full API to this model knowledge base – see subsection 3.4.

3.2 Business Properties

Another concept of ontology-centred modelling implemented in `OpenCASE` are *business properties*. Inspired by the success of this concept in the Craft.CASE tool [3], we implemented a sort of meta-modelling layer enabling to specify custom domain (business) properties. Compared to Craft.CASE we did not limit business properties just to classes of elements (Participant, State, Activity, ...), but we made possible to attach a business property to an element (thus having just one instance) or to an element class (thus having several instances) – Figure 3. Because diagrams are elements as well, they may have business properties orthogonally assigned, too (the author, version, etc.). If we look at states, activities, diagrams, participants and other elements as UML classes, business element class BPs would correspond to class attributes, while element properties would correspond to instance attributes.

3.3 Internal Knowledge Base

The tool is database-centred, i.e. it maintains an internal knowledge base containing functions, scenarios, diagrams, entities, elements together with their graphical properties. Internally, they form nodes of a graph structure and their relations are represented as edges, thus various graph algorithms may be applied on the knowledge base [17]. Graph traversals and graph transformations are probably the most useful and may implement operations like

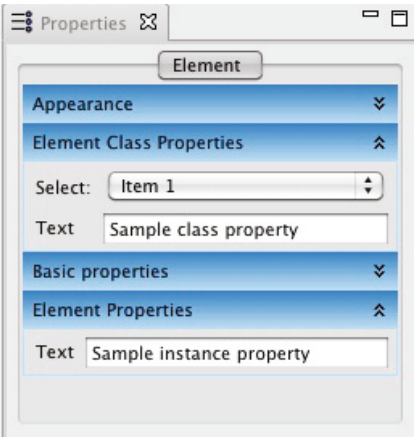

Fig. 3. Element Properties and Element Class Properties in OpenCASE

- *Listings*, like all input/output flows from/to a participant.
- *Calculation of metrics* (like numbers of states and activities in participants) that may be used for *complexity estimations* [16], [15].
- *Calculation of statistics*, e.g. about dataflows and communications (which participants communicate the most/least, above/below average, etc.).
- *Semantics checks*: there is a starting state in every participant, at least one final state[6], ...
- *Conceptual normalisations* [9].
- Any further custom reporting / calculations / processing.

3.4 Model API

We see diagrams as a convenient way how to specify the model and visualize the model to a business user, however the true power lies in its underlying knowledge base. We designed OpenCASE to transparently reveal its API (Application Programming Interface) of the model's structure. Using this API, a programmer may iterate through the model's elements and entities, make verifications, perform various calculations (statistics), transform them to some sort of output (reports) and make inner transformations (e.g. normalisation).

The API is self-documented in the form of UML Class diagrams thanks to the Ecore framework (see section 4). An example of the OR diagram metamodel is in Figure 4.

4 OpenCASE Implementation

OpenCASE is implemented entirely in the Java programming language utilizing the Eclipse Platform and various modelling frameworks from the Eclipse Mod-

[6] According to the BORM method, there may be exceptions to these rules, see [7] for more details.

eling Project (EMP). The Eclipse Rich Client Platform (RCP) offers a very powerful foundation since it provides an extensible component system and a platform for creating complex applications with rich user interfaces.

Due to lack of space in this short paper, we do not provide an overview of the RCP platform. The reader may read about it on the Internet or in the literature – we highly recommend [2], [5], [8], [14].

The core of the `OpenCASE` project is the `OpenCASE` written as an RCP application. There are 10 essential plug-ins constituting the core of the application called `OpenCASE` Workbench. The workbench is just the user interface without any diagram editing capabilities. Diagram manipulation is performed by the remaining plug-ins.

Each feature has a core plug-in having an ID with no suffix, e.g., `org.opencase.diagrams`. Such a plug-in is almost entirely generated from an Ecore model and it implements the basic behaviour of the modelled domain. An example of Ecore model is in Figure 4.

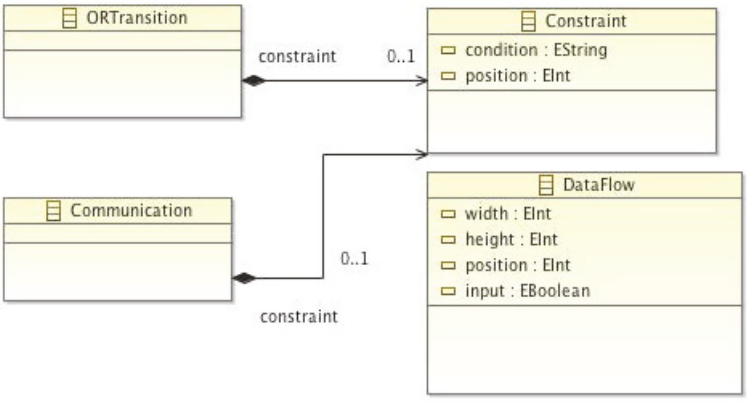

Fig. 4. A part of Ecore model of BORM's Object Relationship Diagram

Currently, there are several plug-ins being developed, compiled and deployed in separation from the core of `OpenCASE`. Thanks to Eclipse plug-in system such components can be installed right into the running `OpenCASE` from an archive or internet update site.

5 Summary, Conclusions and Future Work

`OpenCASE` is an attempt to bring the ontology-centred modelling into everyday life and profit from research achievements while at the same time to provide an open platform for further research. That provided a challenge to implement theoretical results into suitable software implementation and to build a user-friendly tool with features like keyboard shortcuts, complete undo, aligning and distribution of graphical elements, batch operations, etc.

We put a high focus to implement the whole ontology chain, i.e.

1. **Input** – *How to input the terms and their relations in synergy with a concrete notation.* We addressed this issue by separating the identity (entity) from its representation (element) – subsection 3.1
2. **Processing** – *How to access the ontology and manipulate it by transformations and various algorithms* (verifications, normalizations, optimizations, etc.). We built an application programming interface (API) to access the model's knowledge base. The API architecture is documented by UML (Ecore) diagrams[7] – subsection 3.4.
3. **Output** – *How to export the ontological knowledge contained in the model.* Exporters plug-ins handle this task. Exporter plug-ins are implemented as Eclipse plug-ins and they may be implemented to perform an export to various human-readable formats (TXT, HTML, LaTeX, ODT, PDF, ...) or formats suitable for machine processing (CSV, XML, JSON, OWL, ...), or it may perform the export directly into relational database or reveal the knowledge base as a service (SOAP, REST).

At the time of writing this contribution, the modelling core is completely implemented, being further fine-tuned and improved. As for the plug-ins, several output plug-ins are developed (TXT, HTML, LaTeX). We are also working on implementing models simulations and support for optimizations. A huge step toward the holistic ontology-centred conceptual modelling will be implementing other types of diagrams, especially data-structure diagrams (UML Class Diagrams and OntoUML, [6]) and providing a means to make ontologic relations to the process diagrams.

Acknowledgements. This contribution was elaborated with a support of grant no. 20121059 of Grant Agency of The Faculty of Economics and Management of the Czech University of Life Sciences in Prague.

References

1. Brožek, J., Merunka, V., Merunková, I.: Organization Modeling and Simulation Using BORM Approach. In: Barjis, J. (ed.) EOMAS 2010. LNBIP, vol. 63, pp. 27–40. Springer, Heidelberg (2010)
2. Clayberg, E., Rubel, D.: Eclipse Plug-ins. Addison-Wesley Professional (2008)
3. Craft.CASE Tool, `http://www.craftcase.com`
4. Gasevic, D., Djuric, D., Devedzic, V.: Model Driven Engineering and Ontology Development, 2nd edn. Springer (2009)
5. Gronback, R.C.: Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit. Addison-Wesley Professional (2009)

[7] Actually, as was explained, the situation is opposite: the internal structures are *generated* from the Ecore diagrams, which makes a powerful mechanism to maintain specification-implementation consistency. Nevertheless, this is irrelevant for the API's user.

6. Guizzardi, G.: Ontological Foundations for Structural Conceptual Models. Telematica Instituut Fundamental Research Series No. 15 (2005)
7. Knott, R.P., Merunka, V., Polák, J.: The BORM methodology: a third-generation fully object-oriented methodology. Knowledge-Based Systems 16(2), 77–89 (2003)
8. McAffer, J., Lemieux, J.-M., Aniszczyk, C.: Eclipse Rich Client Platform: Designing, Coding, and Packaging Java Applications. Addison-Wesley Professional (2005)
9. Molhanec, M.: Some Reasoning Behind Conceptual Normalisation. In: Information Systems Development, pp. 517–525. Springer Science+Business Media, Berlin (2011)
10. Molhanec, M., Merunka, V.: BORM: Agile Modelling for Business Intelligence. In: Business Intelligence and Agile Methodologies for Knowledge-Based Organizations: Cross-Disciplinary Applications, pp. 120–130. IGI Global, Hershey (2011)
11. MetaEdit+, `http://www.metacase.com/cases/borm.html`
12. Tool, `http://www.opencase.net` (in construction)
13. Pergl, R.: Supporting Enterprise IS Modelling Using Ontological Analysis. In: Barjis, J., Eldabi, T., Gupta, A. (eds.) EOMAS 2011. LNBIP, vol. 88, pp. 130–144. Springer, Heidelberg (2011)
14. Steinberg, D., Budinsky, F., Paternostro, M.: EMF: Eclipse Modeling Framework. Addison-Wesley Professional (2008)
15. Struska, Z., Merunka, V.: BORM points – New concept proposal of complexity estimation method. In: ICEIS 2007: Proceedings of the Ninth International Conference on Enterprise Information Systems: Information Systems Analysis and Specification, 9th International Conference on Enterprise Information Systems (ICEIS 2007), Funchal, Portugal, June 12-16, pp. 580–586 (2007)
16. Struska, Z., Pergl, R.: BORM-points: Introduction and Results of Practical Testing. In: Filipe, J., Cordeiro, J. (eds.) ICEIS 2009. LNBIP, vol. 24, pp. 590–599. Springer, Heidelberg (2009)
17. Valiente, G.: Algorithms on Trees and Graphs. Springer (2010)