# MERLIN:
# an Intelligent Tool for Creating Domain Models

Monique Snoeck✉[0000-0002-3824-3214]

KU Leuven, Research Center on Information Systems Engineering,
monique.snoeck@kuleuven.be

**Abstract.** The complexity of modelling languages and the lack of intelligent tool support add unnecessary difficulties to the process of modelling, a process that is in itself already demanding, given the challenges associated to capturing user requirements and abstracting these in the correct way. In the past, the MERODE method has been developed to address the problem of UML's complexity and lack of formalization. In this paper, we demonstrate how the formalization of a multi-view modelling approach entails the possibility to create smart and user-friendly modelling support.

**Keywords:** conceptual modelling, modelling tool, UML, model-driven engineering, consistency checking.

## 1 Modelling difficulties

Model-driven engineering (MDE) aims to create software from models. One of the key assumptions behind MDE is that models can be made sufficiently complete and correct to generate code from them. While the "general purpose" character of UML makes it widely applicable, it is also its weakness. The UML contains a large number of constructs, making it difficult to use. And when those models are intended to generate code from them, the problem is exacerbated as generating code from a UML model is only possible if the models are sufficiently detailed, which requires a thorough knowledge of the UML. UML is not the only language facing a too large complexity. Several authors have already pointed out that many modelling languages (including the UML) are too “noisy” with various concepts, which inevitably results in creating erroneous models [1, 2]. Also modelling tools are complex, and pose additional challenges to modelers [3], even though good tool support had been proved to lower the likelihood of model quality problems [4].

The MERODE modelling method [5] targets conceptual modelling and addresses these challenges by offering a UML-based modelling approach that allows creating models that conceptual in nature (as opposed to technical designs), yet are sufficiently precise and complete to generate code from them with no more than three clicks. In addition, intelligent tool support has been developed to support the modeller in the best possible way while modelling. The goal of this demo is to demonstrate the features of the MERLIN modelling environment, including multi-view support and intelligent model consistency checking. The modelling environment is freely available since January 2020 at http://www.merlin-academic.com.

## 2 Simplification of the Modelling Language

To ease the use of UML for conceptual modelling, the MERODE method uses a minimal set of concepts from UML required to capture domain models. Three views are supported: structural modelling by means of a UML class diagram, behavioural modelling by means of state charts, and interaction modelling, using a matrix-technique. The three views are directly supported by MERLIN.

In MERODE, the Class diagram only uses the concepts of class, binary association and inheritance associations. More complex concepts such as AssociationClass, composition and aggregation are deliberately not used so as to ease the modellers' task. Moreover, MERODE requires each binary association to be of a one-to-many or of one-to-one type, expressing existence dependency of one object type on the other. Transformation rules from a "classical" UML diagram to such an existence-dependency only class diagram are provided by the method. The following example illustrates the inherent difficulties associated to UML, and how such transformation process results in more precise models, while using a limited number of concepts.

**Fig. 1** identifies three object types[1] (customer, order and salesperson) and two associations: an order is placed by a customer, and is managed by a salesperson. Because both associations are graphically identical, they fail to capture the inherently different semantics of the underlying domain rules: Whereas the customer of an order remains the same for the whole duration of the order, the salesperson managing the order may change over time. In other words, the association end is placed by labelled with a "1" next to customer is not modifiable or frozen, whereas the association end is managed by labelled with a "1" (next to salesperson) is modifiable. **Fig. 1** is not incorrect but is incomplete as it does not allow to discern between frozen and modifiable association ends labelled with multiplicity "1". Depending on the transformation rules, both association ends would either be implemented as modifiable or as frozen, as there is no way to make a distinction based on the UML diagram. In order to obtain a model that captures this difference and can be correctly and automatically transformed to code, we need to express these different semantics by means of different modelling constructs.


**Fig. 1.** Example class diagram

In this case, the MERODE rules dictate to reify the association between order and salesperson as show in **Fig. 2** , thus identifying the concept of 'SalesPersonDuty' as a separate business concept. Reification is also advocated for many-to-many associations, shared aggregation and for composition associations not expressing existence dependency from the part on the whole.

[1] In the remainder of this book we will follow the convention that object types are written in SMALLCAPS and that association names will be underlined.

State charts allow modelling a domain object's behaviour. The UML bases its statechart notation on Harel Statecharts, offering a rich pallet of concepts, including parallellism and decomposition. MERODE however uses only the basic notions: start, intermediate and end states, and requires each class to have just one statechart defining its behaviour.

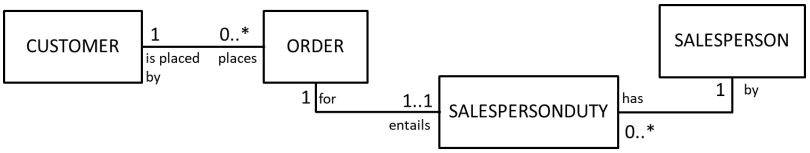


**Fig. 2.** Class diagram with reified association

Likewise, interaction modelling is simplified. In the UML, object interaction aspects are modelled by means of sequence charts and collaboration diagrams, thus forcing a conceptual modeller into premature commitments. In contrast, MERODE follows an event-driven approach that raises events to the same level of importance as objects, and recognizes them as a fundamental part of the structure of experience. In particular, "business events" are captured as the phenomena of interest at the interface between the real world and the information system [6]. An event-object interaction table, inspired from the "CRUD" matrix[2], allows defining the interaction between business events and business objects and using the events as triggers for state transitions in the state charts.

## 3 Intelligent Tool Support

Modelling typically requires describing the same socio-technical system from different perspectives (data, behaviour, authorisations, etc.). Some aspects may however be modelled in more than one scheme, e.g. a business object type appearing in a class diagram and as data object in a BPMN diagram. While many modelling tools focus on one particular diagram, the modelling environment of MERODE allows viewing two models side-by-side, see **Fig. 3**.

Obviously, some kind of consistency checking between different views is required to ensure the quality of the model [7, 8]. This consistency checking can vary from a simple syntactic correspondence to a full semantic match between diagrams. In the past, continuing efforts have been made to provide UML with the needed formal underpinning [9, 10, 11, 12, 13, 14]. Nevertheless, these efforts have not entailed an agreed-upon set of consolidated rules, as a result of which authors define their own set of rules [8]. However, to achieve true consistency, the integration of different views is needed. Recently, the integration between the data and business process perspective is also gaining ground as exemplified by artefact centric approaches [15, 16, 17].

In MERODE, the concept of existence dependency is based on the notion of "life" of an object, and this induces a natural sequencing of creating and ending of objects [14]. A MERODE class diagram is therefore -in spite of its appearance- not a pure

[2] captures how processes **c**reate, **r**ead, **u**pdate or **d**elete data.

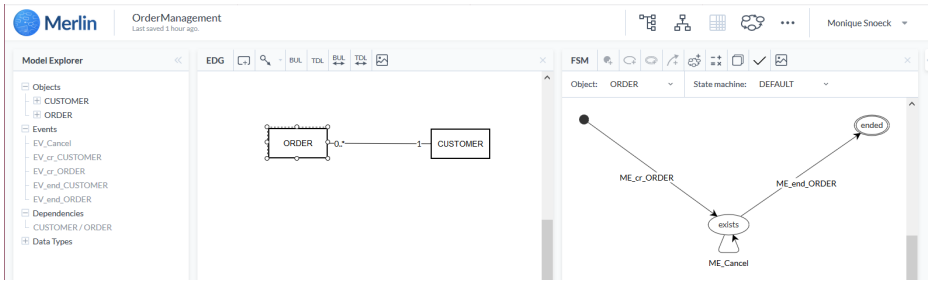


**Fig. 3.** Viewing the class diagram and state charts side by side.

data model, but also defines a default behavioural model. This facilitates consistency checking with other diagrams intended to capture the behavioural aspects of the domain. In contrast with existing tools like Enterprise Architect, Visual Paragdim, etc. that treat the class diagram in a totally independent way from the behavioural models, MERODE defines consistency between the static model and the behavioural model [5] In particular, MERODE sustains three modes of consistency checking [18]:

- *Consistency by construction*, is the most powerful, and means that the tool "auto-completes" model elements based on consistency rules. **Fig. 3**, right, shows the default statechart that is automatically generated for each class, including the generation of default creating and ending events.
- *Consistency by analysis* means that an algorithm is used to detect all potential inconsistencies in the model. The modeller can thus construct the model without caring about temporary incompleteness or contradicting elements. Upon request, a verification algorithm can be run against the models to spot errors and/or incompleteness in the various views. Such verification could be done manually as well, but obviously automated support substantially eases model verification, and is likely to ensure more thorough verification as well. Assume for example, that in an OrderManagement model, the modeller created a state chart like in **Fig. 4**. Running a model checker will then result in a report mentioning the problems of non-determinism and backward inaccessible state shown in **Fig. 5.**

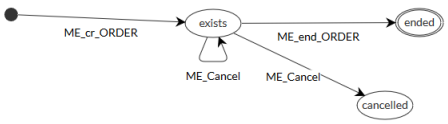


**Fig. 4.** Erroneous state chart

- *Consistency by monitoring*, allows checking new specification against correctness rules when entered in the tool. This allows to maintain the correctness of a model, but it should be used with parsimony as a too stringent verification procedure will turn the input of new model elements into a frustrating activity.

The major advantage of this consistency checking is that this saves a lot of input effort while improving the completeness of the model in one go. Moreover, the auto-

complete functionality avoids the input of inconsistent specifications by completing the entered specifications with their consistent consequences. The result is a much more user-friendly environment. **Fig. 6** shows how model settings in MERLIN allow the user switching on autocomplete functionality, the first of which leads to the automatic creation of creating and ending business events and methods when adding a business object type to the class diagram.

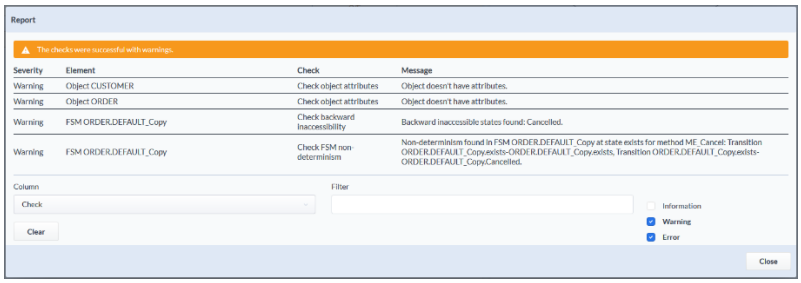


**Fig. 5.** Sample model checking report

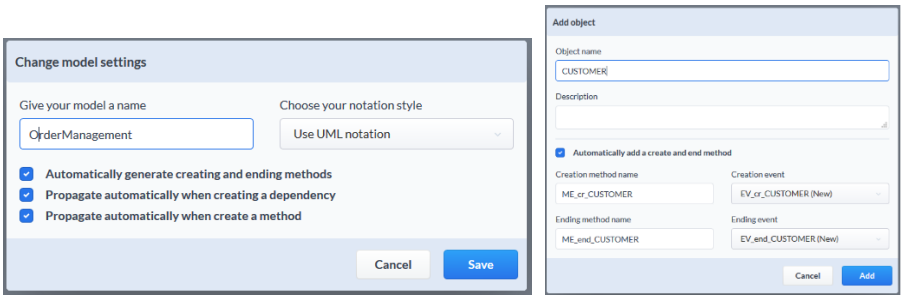


**Fig. 6.** Autocomplete functionality in MERLIN

## 4 Conclusion

The MERLIN modelling environment demonstrates how the formal underpinning of a modelling method, and the formal definition of consistency rules allows offering intelligent and user-friendly modelling support for creating domain models. Besides the benefit for the modeler in terms of modelling effort, the quality of the resulting models entails easy code generation. In the case of MERLIN, the files can be exported to an xml-format allowing the generation of full functional code with a single click. This contributes to model quality too as the generated application is enriched with didactic features helping the modeller to assess the quality of the model [19].

For the purpose of conceptual modelling, MERLIN offers more easy modelling compared to existing tools such as Visual Paradigm and Enterprise Architect: less input is required to achieve a model fit for code-generation, and more consistence checks are offered. On the downside, MERLIN offers a very limited set of modelling constructs, as result of which design choices (such as e.g. navigability of associations, sequence charts) are set by default or part of transformation process. In the future, we plan an XMI-export so that high-level MERLIN-models could be imported in commercial tools to detail the models further.

## References


1. Erickson, J., & Siau, K. (2007). Can UML Be Simplified? Practitioner Use of UML in separate domains. In Proceedings of the 12th Workshop on Exploring Modeling Methods for Systems Analysis and Design (EMMSAD'07), held in conjunctiun with the 19th Conference on Advanced Information Systems (CAiSE'07),Trondheim, Norway (pp.87-96).
2. Wilmont, I., Hengeveld, S., Barendsen, E., & Hoppenbrouwers, S. (2013). Cognitive mechanisms of conceptual modelling. In W. Ng, V. Storey, & J. Trujillo (Eds.), Conceptual modeling (Vol. 8217); (pp. 74e87). Springer Berlin Heidelberg.
3. Siau, K., & Loo, P.-P. (2006). Identifying difficulties in learning uml. Information Systems Management, 23(3), 43-51.
4. Recker, J., et al. How Good is BPMN Really? (2006) Insights from Theory and Practice. in 14th European Conference on Information Systems. 2006. Goeteborg, Sweden: Association for Information Systems.
5. Snoeck, M. (2014) Enterprise Information Systems Engineering: the MERODE approach, Springer
6. Jackson, M. (1995) "The World and the Machine," *1995 17th International Conference on Software Engineering*, Seattle, Washington, USA, 1995, pp. 283-283.
7. Paige, R., Ostroff, J. (2002) "The Single Model Principle", in Journal of Object Technology, vol. 1, no. 5, November-December 2002, pp. 63-81.
8. Torrea, D., Labiche, Y., Genero, M., Elaasar M. (2018) A systematic identification of consistency rules for UML diagrams, Journal of Systems and Software, Vol.1 44, pp. 121-142
9. pUML, The precise UML group, http://www.cs.york.ac.uk/puml/
10. Evans, A., France, R. Lano, K., Rumpe, B., (1998) Developping the UML as a Formal Modelling Notation, in *UML'98 Beyond the notation*; International Workshop Mulhouse France, P-A. Muller, J; Bézivin (eds.),
11. Bruel, J.M., Lilius, J., Moreira A., France R.B, Defining Precise Semantics for UML, ECOOP 2000 Workshop Reaer, LNCS 1964, Springer 2000, pp.113-122.
12. Cheung, K.S., Chow, K.O., Cheung, T.Y. (1998) Consistency analysis on lifecycle model and interaction model, in Proceedings of the International Conference on Object Oriented Information Systems, 9-11 September, Paris
13. Huzar, Z., Kuzniarz, L., Reggio, G., Sourrouille, J.L. (2005) Consistency Problems in UML-Based Software Development. In: Jardim Nunes N., Selic B., Rodrigues da Silva A., Toval Alvarez A. (eds) UML Modeling Languages and Applications. UML 2004. Lecture Notes in Computer Science, vol 3297. Springer, Berlin, Heidelberg
14. Snoeck, M., Dedene, G., Existence Dependency: The key to semantic integrity between structural and behavioral aspects of object types. IEEE Transactions on Software Engineering, 24(24), 233-251.
15. Dumas, M. (2011) On the Convergence of Data and Process Engineering. In: Eder J., Bielikova M., Tjoa A.M. (eds) Advances in Databases and Information Systems. ADBIS 2011. Lecture Notes in Computer Science, vol 6909. Springer, Berlin, Heidelberg
16. Calvanese, D., Montali, M., Patrizi, F., and Rivkin A. (2018). "Modelling and In-Database Management of Relational, Data-Aware Processes." ArXiv:1810.08062 [Cs], October. http://arxiv.org/abs/1810.08062.
17. Künzle, V., Reichert, M. (2011) PHILharmonicFlows: towards a framework for object-aware process management, Journal of Software Maintenance and Evolution: Research and Practice, 23(4) pp. 205-244
18. Snoeck, M., Michiels, C., Dedene, G. (2003). Consistency by construction: The case of MERODE. In: *Lecture Notes in Computer Science*: *vol. 2814*, (105-117).
19. Sedrakyan, G., Snoeck, M., Poelmans, S. (2014). Assessing the effectiveness of feedback enabled simulation in teaching conceptual modeling. Computers and Education, 78, 367-382.