On the Execution of Deep Models

Colin Atkinson, Ralph Gerbig, Noah Metzger
University of Mannheim
Software Engineering Group
Mannheim, Germany
Email: {atkinson, gerbig, metzger}@informatik.uni-mannheim.de

***Abstract*—A variety of tools today support the dynamic execution/simulation of models within a single modeling environment. However, they all suffer from limitations resulting from their implementation on a traditional, two-level modeling platform. The most prominent of these is the inability to represent the specification of the modeling language, the domain model and model execution state at the same time in a uniform and seamless manner. They therefore invariably have to resort to some kind of ad hoc extension mechanism or workarounds to represent all three levels, with corresponding increases in accidental complexity and potential for misunderstandings. In this paper we demonstrate how deep modeling environments provide a conceptually cleaner and more powerful environment for model execution and simulation thanks to their inherent support for the representation of arbitrary numbers of classification levels, and the ability to define customizable, domain specific languages within them.**

I. Introduction

Modeling languages and tools aiming to support the execution of models are available in various different flavors and degrees of maturity. Some tools support the graphical definition of executable models (e.g. fUML [1] in xMOF [2] or graph transformations in AToMPM [3]). Others support the textual definition of execution semantics (e.g. ALF [4]). Sophisticated, industry-quality simulation environments such as Simulink [5] or AnyLogic [6] are also available, with accompanying languages.

All these languages and tools are based on traditional "two-level" modeling technology which limits the modeler to two "physical" levels for modeling - one containing types and one containing instances of those types. Usually the type level is used to define the modeling language (meta-level) and the instance level is used to represent the user model (e.g. UML class or state diagrams). However, with such an arrangement there is no place left to model the instances of the user model which is where the majority of the execution actions conceptually take place. In general, at least three levels are needed to naturally represent model execution in a modeling tool. To address the lack of sufficient modeling levels in traditional OMG-based technologies, workarounds are needed to show run-time data at the level of the executed user model (i.e. execution blueprint).

Deep modeling provides a natural solution to this problem by providing uniform, "out-of-the-box" support for modeling over multiple classification levels. As well as supporting the extra classification level required to store execution information. Deep modeling also provides a number of additional advantages. First, it makes it possible to naturally represent run-time instance information alongside model type information (with inherent support for the semantics of instantiation) without polluting the model execution blueprint. Second, it provides natural support for debugging by allowing model execution traces (i.e. instances) to be directly checked against model execution blueprints (i.e. types). Third, it allows the behavior of the system to be dynamically modified at any time by simply changing model execution instances without changing the blueprint. Fourth, the execution blueprint can be extended dynamically at any time, without the need for any code generation or editor redeployment, simply by adding new types to the language definition. Finally, it allows domains that inherently feature more than three classification levels to be modeled and executed in a natural and uniform way since there is no limitation on the number of levels that can be modeled.

In this paper we illustrate the advantages of the deep modeling approach for model execution by presenting an example of the execution of a simple but intuitive deep model from the domain of gaming. The example represents an executed game in which, from a birds-eye view, two players play against each other on one computer. One uses the mouse to control the game and the other the keyboard. Changes to the state of the game (i.e. the deep model) are immediately visible to all parties, and a third player is able to manipulate the game while the other two players play.

The paper is structured as follows: in the next section (Section II) the deep modeling approach is introduced. The prototype game demonstrating the advantages of deep models for model execution is then shown in Section III and discussed in Section IV. Finally the paper closes with a discussion of related work (Section V) and a few concluding remarks (Section VI).

II. Deep Modeling

Deep modeling environments allow domains with more than one logical class-/instance level to be represented within one physical model. In the domain of model simulation and execution at least three levels are usually required — one defining the modeling language, one describing "the model" and one capturing the execution state of the model in the form of instances. Figure 1 shows an excerpt of the deep model used for modeling the executed game environment. Although this model has only three levels (i.e. O_0, O_1, O_2) the number of levels is unlimited in general.

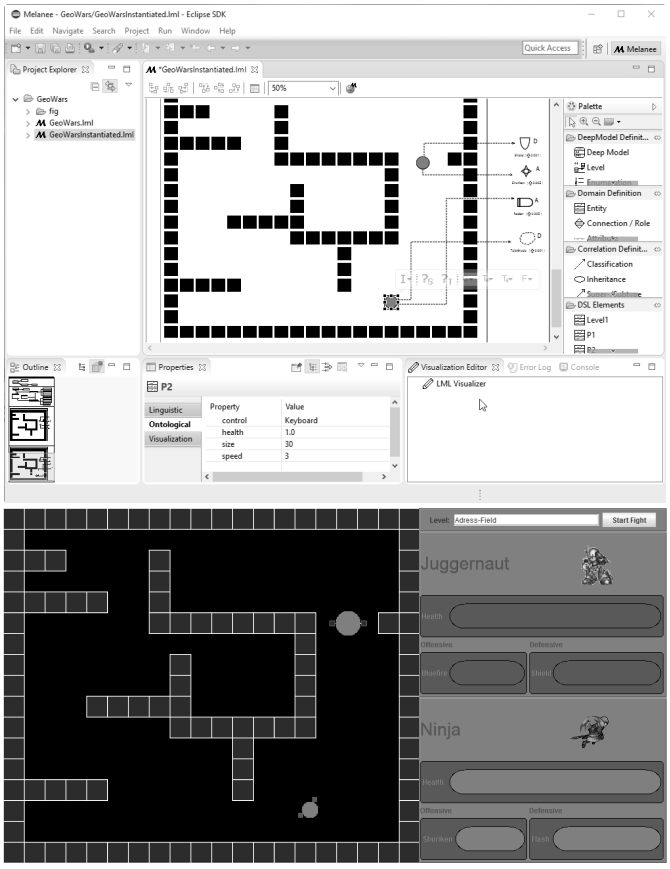

Fig. 4. The GeoWars example running in Melanee and the GeoWars Engine.

level executed by the game engine is shown at the bottom of the figure. A video of the running tool can be found on the Melanee homepage [11] while the game and source are available at [11] and [12].

IV. Discussion

This GeoWars game example shows that it is possible to naturally implement sophisticated model execution environments using deep modeling tools such as Melanee. Melanee provides all the capabilities required for a model execution scenario including: 1) services for model query and manipulation, 2) services for accessing and manipulating graphical model representation definitions, 3) extensibility via a plug-in framework supporting a general purpose programming language (Java) and 4) support for defining execution semantics through action languages and transformations. In areas where the Melanee API is not powerful enough the well-established technologies of the Eclipse Platform can be used as a fallback. The tools ongoing use for other projects allows missing features to be continuously discovered and added to the Melanee modeling environment and made available via the Melanee API.

The model execution scenario shown here can also be implemented with traditional modeling technologies available today. These "two-level" modeling technologies provide a type level which would contain the GeoWars modeling language located at O_1 in Figure 3 and an instance level which would contain the executed level blueprint which is located at level O_2 in Figure 3. The execution information is then displayed as additional information in the model execution blueprint (i.e. the model game level) by applying workarounds such as UML Profiles or the annotation model approach [13] because a dedicated classification level for representing execution information is not available in such a modeling approach.

From a conceptual point of view the deep modeling approach is much cleaner than model execution approaches based on the aforementioned traditional "two-level" modeling approach. In particular, it naturally includes all the classification levels needed to represent the currently executed model. Changes to a running model execution can be defined at the level of a specific model instance to influence the state and behavior of that one particular game only. The model instances can also be used to pause and resume model executions. Such a stack of classification levels is not available in "two-level" modeling technologies and thus data about the state of the executed model has to be saved externally outside the modeling stack. Moreover, modifications to the executed model on a per-instance basis cannot be defined without applying workarounds. In such an architecture, changes can only be performed at the blueprint level which then effects all executions of the model and not a single instance only.

Visualizing the current state of an executed model is also typically done at the blueprint level rather than at the instance level today. This highlights the conceptual problem faced in the representation of executing models within a two-level environment as the user gets the impression that the blueprint is executed instead of a model instance. Deep modeling on the other hand supports the visualization of model instances during model execution and leaves the blueprint untouched.

To summarize, the example demonstrates that deep modeling naturally supports the key features needed for model execution which are: 1) the availability of additional classification levels dedicated to executed model instances, 2) the presentation of the current execution state of a system at the instance level, 3) storing instance information of executed models in the model itself and 4) level-agnostic action and transformation languages to define execution semantics. All of these features are available in today's tools but as extensions to, and workarounds on, the underlying modeling approach which is not optimal for the tasks previously mentioned. Unnatural extensions and workarounds increase the accidental complexity [14], [15] of models.

V. Related Work

Three different areas of work are relevant to the technology presented in this paper: deep modeling tools with model execution capabilities, standard modeling tools with built-in execution/simulation support and approaches to connect pure modeling tools with third party execution/simulation tools.

The only deep modeling tool besides Melanee which allows execution of models is Metadepth [16] which is shipped with

an action language from the Eclipse Epsilon Framework [17]. However, MetaDepth, itself only provides textual visualizations which limits the kind of applications that it can support. The example shown in this paper is mainly graphical but Melanee can support multi-format visualizations (e.g. text, forms, tables).

In the two-level modeling space, Atom3 [18] and its successor AtomPM [3], are the best known academic tools with simulation and execution capabilities. The best known industrial tools are AnyLogic [6] and Simulink [5]. AtomPM allows a model to be simulated/executed by specifying a graph transformation which is much more intuitive and easier to realize than writing a plug-in for the Melanee tool. On the other hand, we believe that this approach is less flexible and powerful when it comes to complex computations or the use of third party components as shown in this paper. One of the most advanced academic tools supporting model execution is xMOF [2], which operates on fUML [1] specifications. However, because of its exclusive focus on UML it cannot be applied to arbitrary domain-specific languages unlike Melanee.

An approach for connecting modeling tools with external execution capabilities is described by Fritzsche et. al. [19]. They present an approach which allows simulation information to be attached to any model and subsequently exported and simulated/executed in another external tool. The model can then be updated by importing the simulation/execution trace back into the modeling tool. However, this approach adds a lot of extra effort to modify existing modeling environments if an integrated tool experience is desired.

VI. Conclusion

This work shows the advantages of deep modeling for supporting the simulation and execution of models. The prototype implementation shows that even the prototype deep modeling tools available today are already up to the task. In the paper we described the conceptual shortcomings of current modeling technologies and explained how deep modeling overcomes them. The shortcomings mainly revolve around the inability of traditional "two-level" modeling environments to naturally support more than one type and one instance level. The proposed deep modeling approach addresses this limitation through 1) the availability of additional classification levels dedicated to executed model instances, 2) the presentation of the current execution state of a system at the instance level, 3) storing instance information of executed models in the model itself and 4) level-agnostic action and transformation languages to define execution semantics. We are continually enhancing the capabilities of the Melanee deep modeling tool and hope the example in this paper will encourage other researchers to investigate the benefits of this technology for model simulation and execution.

References

[1] OMG, "Semantics of a foundational subset for executable uml models (fuml)," http://www.omg.org/spec/FUML/1.1/, 2013.

[2] T. Mayerhofer, P. Langer, M. Wimmer, and G. Kappel, "xmof: Executable dsmls based on fuml," in *Software Language Engineering*, ser. Lecture Notes in Computer Science, M. Erwig, R. Paige, and E. Van Wyk, Eds. Springer International Publishing, 2013, vol. 8225, pp. 56–75. [Online]. Available: \url{http://dx.doi.org/10.1007/978-3-319-02654-1_4}

[3] E. Syriani, H. Vangheluwe, R. Mannadiar, C. Hansen, S. V. Mierlo, and H. Ergin, "Atompm: A web-based modeling environment," in *Joint Proceedings of MODELS'13 Invited Talks, Demonstration Session, Poster Session, and ACM Student Research Competition co-located with the 16th International Conference on Model Driven Engineering Languages and Systems (MODELS 2013)*, vol. 1115, 2013, pp. 21–25.

[4] E. Seidewitz, "Uml with meaning: Executable modeling in foundational uml and the alf action language," *Ada Lett.*, vol. 34, no. 3, pp. 61–68, Oct. 2014. [Online]. Available: http://doi.acm.org/10.1145/2692956.2663187

[5] J. B. Dabney and T. L. Harman, *Mastering simulink*. Pearson, 2004.

[6] A. Borshchev, *The Big Book of Simulation Modeling: Multimethod Modeling with AnyLogic 6*. AnyLogic North America, 2013.

[7] C. Atkinson, M. Gutheil, and B. Kennel, "A flexible infrastructure for multilevel language engineering," *Software Engineering, IEEE Transactions on*, vol. 35, no. 6, 2009.

[8] C. Atkinson and R. Gerbig, "Melanie: Multi-level modeling and ontology engineering environment," in *Proceedings of the 2nd International Master Class on Model-Driven Engineering: Modeling Wizards*, ser. MW '12. New York, NY, USA: ACM, 2012, pp. 7:1–7:2.

[9] ——, "Aspect-oriented concrete syntax definition for deep, user-defined modeling languages," ser. MULTI'15, 2015.

[10] A. Kleppe, *Software Language Engineering: Creating Domain-Specific Languages Using Metamodels*, 1st ed. Addison-Wesley Professional, 2008.

[11] Melanee, "Project Website," http://www.melanee.org, 2015.

[12] ——, "Source Code," https://melanee2.informatik.uni-mannheim.de/stash, 2015.

[13] M. Fritzsche, J. Johannes, U. Aßmann, S. Mitschke, W. Gilani, I. Spence, J. Brown, and P. Kilpatrick, "Systematic usage of embedded modelling languages in automated model transformation chains," in *Software Language Engineering*, ser. Lecture Notes in Computer Science, D. Gasevic, R. Lämmel, and E. Van Wyk, Eds. Springer Berlin Heidelberg, 2009, vol. 5452.

[14] J. Brooks, F.P., "No silver bullet essence and accidents of software engineering," *Computer*, vol. 20, no. 4, pp. 10–19, 1987.

[15] C. Atkinson and T. Khne, "Reducing accidental complexity in domain models," *Software & Systems Modeling*, vol. 7, no. 3, pp. 345–359, 2008. [Online]. Available: http://dx.doi.org/10.1007/s10270-007-0061-0

[16] J. de Lara and E. Guerra, "Deep meta-modelling with metadepth," in *Proceedings of the 48th international conference on Objects, models, components, patterns*, ser. TOOLS'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 1–20.

[17] D. S. Kolovos, R. F. Paige, and F. A. C. Polack, "The epsilon object language (eol)," in *Model Driven Architecture Foundations and Applications*, ser. Lecture Notes in Computer Science, A. Rensink and J. Warmer, Eds. Springer Berlin Heidelberg, 2006, vol. 4066, pp. 128–142. [Online]. Available: http://dx.doi.org/10.1007/11787044_11

[18] J. Lara and H. Vangheluwe, "Atom3: A tool for multi-formalism and meta-modelling," in *Fundamental Approaches to Software Engineering*, ser. Lecture Notes in Computer Science, R.-D. Kutsche and H. Weber, Eds. Springer Berlin Heidelberg, 2002, vol. 2306, pp. 174–188.

[19] M. Fritzsche, "Performance related Decision Support for Process Modelling," Ph.D. dissertation, Queen's University Belfast, 2010.