# Integration of Runtime Verification into Metamodeling

F. Macías[1], T. Scheffel[2], M. Schmitz[2], R. Wang[1],
M. Leucker[2], A. Rutle[1], and V. Stolz[1]

[1] Bergen University College, Norway
{fernando.macias, rui.wang, volker.stolz, adrian.rutle}@hib.no
[2] Institute for Software Engineering and Programming Languages,
University of Lübeck, Germany
{scheffel, schmitz, leucker}@isp.uni-luebeck.de

## 1 Domain Specific Modeling Languages (DSMLs)

Modeling is a well-established practice in the development of big and complex software systems. Domain Specific Modeling Languages (DSMLs) are a technique used for specifying such systems in an abstract way. These languages define the structure, semantics and constraints for models related to the same application domain. The models created with DSMLs are then interconnected or related to one another. Among the reasons for tailoring a language to the problem space is their better understandability by domain experts, capacity for high-level abstraction, and user friendliness. However, the use of DSMLs (like the use of types in general) does not shield the produced software from bugs or man-made mistakes. Software failures may still occur on complex systems due to a variety of reasons such as design errors, hardware breakdown or network problems. Ruling out these failures requires that verification methods that guarantee correct execution even in corner cases are integrated into the development process.

We previously presented our ideas for integrating specifications tighter with the model [4]. We improve on our realisation in the context of behavioural models for embedded systems [6], and now tackle modelling and specification of a distributed system. LEGO MindStorm® robots serve as the platform for a small case study.

## 2 Runtime Verification (RV)

The use of verification methods during the specification of a system can greatly improve their reliability. A commonly used verification technique is testing. Unfortunately, testing is seldom exhaustive and cannot always guarantee correctness. An exhaustive option to check every execution path is model checking. But this alternative may suffer the state space explosion problem [3], especially relevant in distributed systems due to their inherent non-determinism. Yet another possibility in the system verification domain is to use runtime verification (RV). Runtime verification is an approach growing in popularity to verify the correctness of complex and distributed systems by monitoring their executions. It can cope with the inadequacies of testing by reacting to systems' failures as soon as they occur. Also, it is a much more lightweight technique when compared to model checking, since only one execution path is checked. Generally, RV can be used to check whether the current execution of the complex system violates given correctness properties [3]. Such checking can be typically performed and decided by using a monitor which, in the simplest form, outputs either true or false.

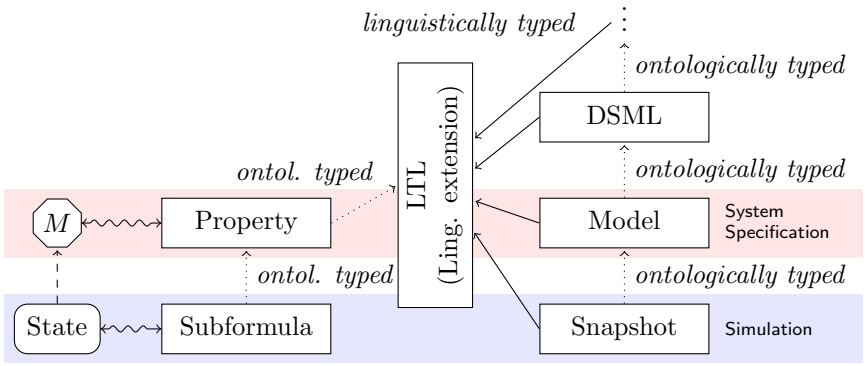


Figure 1: Multilevel model hierarchy with linguistic extension.

# 3 Combining RV and DSMLs

Our research idea is to integrate RV and domain specific modeling into the development process of complex systems. Such integration is achieved by linking the elements of the system model with the atomic propositions of the temporal correctness properties used to specify monitors. In order to achieve this idea, we view a system as having a state consisting of a set of atomic propositions. In RV, we specify correctness properties based on the atomic propositions and generate monitors from them. With this, monitor statements about the correctness of the current execution of the system can then be made. To be compatible with this view of a system, we define a multilevel modeling hierarchy [1] where the DSML and the actual model of the system are included. Moreover, the hierarchy includes instances of the system model that represent the particular state of the system at a given point in time. We call the instances of the system model *snapshots*. A snapshot is also a model, and contains the set of active elements of the system. The way in which the system evolves during the simulation is described using model transformations that generate a new snapshot from the previous one. In short, to link both RV and DSMLs, we associate the atomic propositions, used in RV, with the current state of the system, represented as a snapshot through execution or simulation of a domain specific model. Fig. 1 shows the different modeling levels and their elements in relation to each other. From a metamodeling perspective, every model on the right-hand side of the figure is *ontologically typed* by the model on top of it. The relation with the linguistic extension in the center of the picture is established by *linguistic typing*, that allows to give a second type to every element, orthogonal to the previous one [7]. Our implementation of a multilevel metamodelling tool that supports these concepts is described in [5] and available from `http://prosjekt.hib.no/ict/multecore/`.

## 3.1 Linking a DSML with temporal properties

Additionally, together with an example DSML, we also define the integration of its behavioral semantics with the evaluation of temporal properties. We achieve such integration by linking the elements of the DSML with the atomic propositions used in Linear Temporal Logic (LTL) formulas. We implement the syntax of a temporal logic as a linguistic extension. The key concept of a linguistic extension for our work is the possibility to connect *any* type of element or set of elements in the modeling hierarchy to the model representing the temporal property. We connect single elements in the snapshots to the temporal properties: an atomic proposition

in the LTL property is a fragment of a model instance that may appear in a snapshot. The atomic propositions are evaluated as follows: if at least one match of the fragment appears in the current snapshot of the system, the atomic proposition is evaluated to true, otherwise to false. This connection of elements and atomic propositions allows us to look at the sequence of snapshots of a system as the system's execution, from both RV and modelling points of view. So we can do RV with temporal logics in a natural way based on those snapshots, because they represent the states of the system during the execution. By doing so the connection between the correctness property and the modeled behavior of the system is kept consistent throughout the software engineering process, as propositions always refer to existing model elements.

### 3.2 Extensions

To make our approach more widely applicable, we present a detailed metamodel that captures a wider range of aspects of the robots (sensors, motors, communication), and allows design of distributed systems. Distribution can be modeled through replication of existing instances on the modeling level, or through composition. Upon deployment, code generated for a robot in the model is mapped to the corresponding physical entity.

As the property specification language spans the entire model, this gives naturally rise to partial monitors in each system that contribute to a global property. Annotations structure the evaluation of the sub-formulas into a hierarchy with an explicit mapping to individual instances. Code generation for the runtime monitors takes this into account and yields partial monitors using the ideas of Distributed Temporal Logic (DTL, [8]).

The connection of the atomic propositions to the model elements (states, sensors and motors) enables us to expand our approach to different LTL semantics like $\text{LTL}_3$ [2] in order to report final fulfillment or violation of the correctness properties as soon as possible when monitoring. Timed LTL would allow us to express real-time properties, and we can add other theories for using variables instead of only boolean propositions.

## References


[1] Colin Atkinson and Thomas Kühne. Reducing accidental complexity in domain models. *Software & Systems Modeling*, 7(3):345–359, 2008.

[2] Andreas Bauer, Martin Leucker, and Christian Schallhart. Runtime Verification for LTL and TLTL. *ACM Trans. Softw. Eng. Methodol.*, 20(4):14:1–14:64, 2011.

[3] Martin Leucker and Christian Schallhart. A brief account of runtime verification. *The Journal of Logic and Algebraic Programming*, 78(5):293–303, 2009.

[4] Fernando Macías, Adrian Rutle, and Volker Stolz. A Property Specification Language for Runtime Verification of Executable Models. In *Nordic Workshop on Programming Theory (NWPT)*, pages 97–99, 2015. Tech. Rep. RUTR-SCS16001, School of Computer Science, Reykjavik University.

[5] Fernando Macías, Adrian Rutle, and Volker Stolz. MultEcore: Combining the best of fixed-level and multilevel metamodelling. In *3rd Intl. Workshop on Multi-Level Modelling*, CEUR Workshop Proceedings. CEUR-WS.org, 2016. To appear.

[6] Fernando Macías, Torben Scheffel, Malte Schmitz, and Rui Wang. Integration of runtime verification into meta-modeling for simulation and code generation. In *Intl. Conf. on Runtime Verification (RV'16)*, LNCS. Springer, 2016. To appear.

[7] Alessandro Rossini, Juan de Lara, Esther Guerra, Adrian Rutle, and Uwe Wolter. A formalisation of deep metamodelling. *Formal Aspects of Computing*, 26(6):1115–1152, 2014.

[8] Torben Scheffel and Malte Schmitz. Three-valued asynchronous distributed runtime verification. In *Formal Methods and Models for Codesign, MEMOCODE*, pages 52–61. IEEE, 2014.