Toward Megamodels at Runtime

Thomas Vogel, Andreas Seibel, and Holger Giese

Hasso Plattner Institute at the University of Potsdam
Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam, Germany
prename.surname@hpi.uni-potsdam.de

Abstract. In model-driven software development a multitude of development models that are related with each other are used to systematically realize a software system. This results in a complex development process since these models and the relations between these models have to be managed. Similar problems appear when following a model-driven approach for managing software systems at runtime. A multitude of runtime models that are related with each other are likely to be employed simultaneously, and thus they have to be maintained at runtime. While for the development case *megamodels* have emerged to address the problem of managing development models and relations, the problem is rather neglected for the case of runtime models by applying ad-hoc solutions. Therefore, we propose to utilize concepts of megamodels in the domain of runtime system management. Based on existing work in the research field of runtime models, we demonstrate that different kinds of runtime models and relations are already employed simultaneously in several approaches. Then, we show how megamodels help in structuring and maintaining runtime models and relations in a model-driven manner while supporting a high level of automation. Finally, we present two case studies exemplifying the application and benefits of megamodels at runtime.

1 Introduction

According to France and Rumpe, there are two broad classes of models in *Model-Driven Engineering* (MDE): *development models* and *runtime models* [1]. Development models are employed during the model-driven development of software. Starting from abstract models describing the requirements of a software, these models are systematically transformed and refined to architectural, design, and implementation models until the source code level is reached.

In contrast, a runtime model provides a view on a running software system that is usually used for managing the system at runtime. Therefore, a runtime model serves as a basis and interface for monitoring, analyzing, and adapting a running system, which is realized by causally connecting the model and the system [1, 2]. Most approaches, like [3–6], employ *one* causally connected runtime model that reflects a running software system. While it is commonly accepted that developing complex software systems using *one* development model is not practicable, we argue that the whole complexity of managing a running software system cannot be covered by one runtime model defined by one metamodel. This is also recognized by Blair et. al who state "that in practice, it is likely that

multiple [runtime] models will coexist and that different styles of models may be required to capture different system concerns" [2, p.25].

At the *2009 Workshop on Models@run.time* we presented an approach for using multiple runtime models at different levels of abstraction simultaneously for monitoring and analyzing a running system [7]. Each runtime model defined by a different metamodel abstracts from the system and focuses on a specific concern, like architectural constraints or performance. At the workshop, our approach raised questions and led to a discussion about simultaneously coping with these models since concerns that potentially interfere with each other are separated in different models [8]. For example, any adaptation being triggered due to the performance state of a running system, which is reflected by one runtime model, might violate architectural constraints being reflected in a different model. Thus, there exists relations, like trade-offs or overlaps, between different concerns or models, which have to be considered for runtime management.

A similar issue appears during the model-driven development of software. A multitude of development models and relations between those models have to be managed. An example is the *Model-Driven Architecture* (MDA) approach that considers, among others, transformations of platform-independent to platform-specific models [9]. Thus, different development models are related with each other, and if changes are made to any model, the related models have to be updated by synchronizing these changes or by repeating the transformation. In this context *megamodels* have emerged as one means to cope with the problem of managing a multitude of development models and relations. The term megamodel is known since Jean Bézivin et al. and Jean Marie Favre published their ideas on modeling MDA and MDE, respectively [10, 11]. Both authors basically agree that *a megamodel is a model that contains models and relations between those models or between elements of those models* (cf. [10–13]).

In contrast, the problem of managing multiple models and relations is neglected for the runtime case and to the best of our knowledge there is no approach that explicitly considers this problem beyond ad-hoc and code-based solutions. In this paper, we present categories of conceivable runtime models and possible relations between those models. Based on that, we propose to apply existing concepts of megamodels for managing runtime models and relations. Such an approach provides a high level of automation for organizing and utilizing multiple runtime models and their relations, which supports the domain of runtime system management, e.g., by automated impact analyses across related models.

The rest of the paper is structured as follows. Section 2 discusses the categorization of runtime models and relations. Section 3 describes the application of megamodels at runtime, which is exemplified by two case studies in Section 4. Finally, the paper concludes and gives an outlook on future work in Section 5.

2 Runtime Models and Relations Between Them

In this section, we present categories of conceivable runtime models and relations between them based on the current state of the research field, primarily the past *Models@run.time* workshops [14] and our own work [7, 15–17]. However, we do

not claim that the presented categories are complete or that each category has to exist in every approach. Nevertheless, they indicate that different kinds of runtime models are likely to be employed simultaneously and that these models themselves together with their relations have to be managed at runtime.

2.1 Categories of Runtime Models

Each of the already mentioned approaches [3–6] employs one runtime model reflecting the running system. In contrast, our approach [7] provides multiple runtime models simultaneously, each of which reflects the running system and is specified by a distinct metamodel. Nevertheless, almost all of the other approaches also maintain additional model artifacts at runtime. These artifacts do not reflect the running system, but they are used for runtime management.

In the case of *Rainbow* [6], such artifacts are invariants that are checked on the runtime model, and adaptation strategies that are applied if invariants are violated. Morin et al. [4] even have in addition to an architectural runtime model reflecting the running system, a feature model describing the system's variability, a context model describing the system's environment, and a so called reasoning model that can take the form of event-condition-action (ECA) rules describing which feature should be (de-)activated on the architectural model depending on the context model. Thus, even if only one causally connected runtime model is used for reflecting the running system, several other models are employed at runtime. For the following categories as depicted in Figure 1, we consider any conceivable *Runtime Models* regardless whether they reflect a running system or not. The models are categorized according to their purposes and what they represent. Runtime models of all categories are usually instances of *Runtime Metamodels* conforming to *Runtime Meta-Metamodels*, which leverages typical MDE techniques, like model transformation or validation, to the runtime.

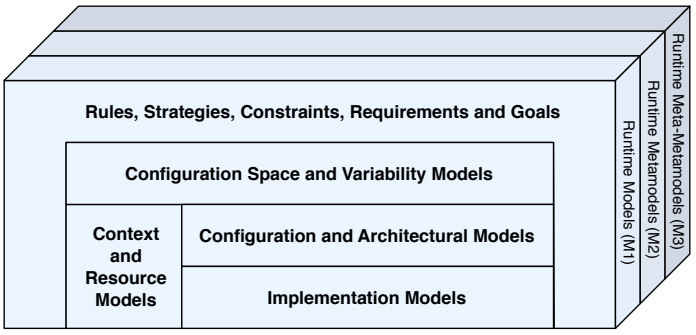

Fig. 1. Categories of Runtime Models

Implementation Models are similar to models used in the field of reflection to represent and modify a running system through a causal connection. Thus, these models are dynamic as they evolve consistently with a running system. Such models are based on the solution space of a system as they are coupled to the system's implementation and computation model [2]. Examples are models used in reflective programming languages, which represent the building blocks of

the languages [18, 19], or models that are directly coupled to platforms or technologies like *CORBA* [20]. Therefore, these models are rather platform-specific and at a low level of abstraction. Examples of such models are class or object diagrams and scenario-based sequence diagrams covering the interaction between objects or generally traces of a system [18, 21, 22]. Moreover, behavioral models in the form of statecharts, state machines, or generally automatons are used to reflect the current state of objects or of a running system [23–25].

Configuration and Architectural Models are at a higher level of abstraction than *Implementation Models*, but they usually also provide causally connected representations of running systems. Such a model reflects the current configuration of a system and it is the core model for monitoring and adapting the system. Since software architectures are considered as an appropriate abstraction level for performing adaptations, such models often provide architectural views on a running system [3–7, 17]. Thus, these models are often similar to component diagrams, which are often annotated or enhanced with elements or attributes to address non-functional properties, like performance or reliability [6, 7]. Therefore, these models are also the basis for analysis either by directly performing the analysis on them or by transforming them to specific analysis models, like queuing networks in the case of performance management. At a even higher level of abstraction, process or workflow models are also feasible to describe a running system from a business-oriented view [26]. Moreover, model types of the *Implementation Models* category, like statecharts or sequence diagrams, are also conceivable in this category, but at a higher level of abstraction. For example, a sequence diagram would consider the interactions between component instances instead of the interactions between objects.

In general, models of this category are rather related to problem spaces and they abstract from the implementation models and from underlying technologies to provide platform-independent views. This corresponds to the view of Blair et al. [2] on runtime models. With respect to a self-adaptive system, these models enable the self-awareness of the system at an appropriate level of abstraction, which is used as a basis for the feedback loop, i.e., for monitoring and analyzing the system, and for planning and executing adaptations on the system.

Context and Resource Models describe the operational environment of a running system. This comprises the context of a system, which is "any information that can be used to characterise the situation of an entity", while "an entity is a person, place, or object that is considered relevant to the interaction between a user and an application" [27, p.5] or in general to the operation of the application. Especially for a context-aware system, which is a system that adapts its behavior to changes in its context, the context has to be observed by sensors and described by a model. A simple example for a context is the user's location, which can be used in mobile systems to find services, like restaurants, in the vicinity of the user. To represent a context, a variety of models can be used: semi-structured tags and attributes, object-oriented or logic-based models [28], or some form of variables, like key value pairs [4, 28]. Even feature models have been proposed for modeling context [29].

Moreover, the operational environment consists of resources a running system requires for operation. These are logical resources, like any form of data, or physical resources, like hardware. An example for a resource model reflects the hardware infrastructure, like computing nodes and network links among nodes, on which the system is running. Therefore, such a resource model provides information whether any adaptation of a system is feasible based on the currently provided resources, like on which node a subsystem can be deployed.

Configuration Space and Variability Models specify potential variants of a running system, while *Configuration and Architectural Models* reflect the currently running variant of the system. Therefore, models of this category describe a system at the type level to span the system's configuration space and variability. Considering a component-based system, the configuration space is defined by the available types of components that can be instantiated and deployed to a running system. Thus, adaptation points in a running system and possible adaptation alternatives can be identified using these models.

Examples for models in this category are component type diagrams [16, 17], feature models originating from dynamic software product lines [4, 30, 31], or aspect models describing variants of a system and instances of these aspects are woven into configuration or architectural models for adapting the system [4, 32].

Rules, Strategies, Constraints, Requirements and Goals may refer to any model from the other categories and, therefore, their levels of abstraction are similar to the levels of the referred models. However, considering requirements or goals at runtime aims at higher levels of abstraction, even above the level of software architectures [33]. Models in this category define, among others, when and how a running system should be adapted. According to Fleurey and Solberg [34] there are two general approaches to specify adaptations. First, adaptation rules or reconfiguration strategies usually in some form of ECA rules describe when (periodically or at the occurrences of context or system events) and under which conditions, a system is adapted by performing reconfiguration actions. The second approach is based on goals a running system should achieve, and adaptation aims at optimizing the system with respect to these goals. This optimization process is based on utility functions to find the best or at least an appropriate target system configuration fulfilling the goals. Both approaches use models reflecting the current system, context, and resources to search the configuration space for a variant that is appropriate for the current state. For example, a goal-based optimization model is used in [4, 31, 35], and adaptation rules or reconfiguration strategies are used in [6, 29, 36].

Moreover, constraints on models of the other categories regarding functional and non-functional properties are used for runtime validation and verification purposes, and for guiding adaptations. If a constraint is violated, an adaptation can be triggered, as it is done in [6], or constraints may exclude certain kind of adaptations. Constraints can be expressed, among others, in the *Object Constraint Language* (OCL), like in [7] to check architectural constraints, or formally in some form of *Linear Temporal Logic* (LTL), like in [23] to verify adaptive systems at runtime. Though constraints can be seen as requirements that

are checked at runtime, recently the idea of *requirements reflection* has emerged, which explicitly considers requirements as adaptive runtime entities [33]. Thus, requirements models, like goal models, become runtime models that have to be related to *Configuration and Architectural Models* since any changes at the requirements level have to be reflected in the running system, and vice versa.

The presented categories show that different kinds of runtime models are possible and even employed simultaneously. Which categories are used, and which kind of and how many models for each of the used categories are employed is specific to each approach. This depends, among others, on the purposes of an approach (which functional and non-functional concerns are of interest, which management activities, like monitoring, analysis or adaptation, are supported, etc.) and on the domain of the managed system (embedded, mobile, or server-side systems, or even IT infrastructures, etc.). Based on the model categories, conceivable relations between runtime models are presented in the next section.

2.2 Relations Between Runtime Models or Model Elements

In the following, we outline exemplars of relations between runtime models to motivate the need for runtime management of relations together with the models.

As already mentioned, models of the category *Rules, Strategies, Constraints, Requirements and Goals* may refer to models of the other categories. For example, goal modeling approaches refine a top-level goal to subgoals recursively until each subgoal can be satisfied by an agent being a human or a software component [37]. Having a goal model at runtime, it is of interest which component of a running system actually satisfies or fails in satisfying a certain goal. Therefore, goals being reflected in a goal model refer to corresponding components of *Configuration and Architectural Models* such that a goal model and an architectural model are related at runtime. Moreover, goal satisfaction can be influenced by the current context of a system, such that goals and elements of a context model and, therefore, the goal model and *Context Model* are related with each other.

Another exemplar describes an instance-of relation between *Configuration Models* and *Configuration Space Models*. For example, a configuration space is defined by the types of available components and an actually running system consists of instances of these types. At runtime, this relation is useful for navigating from configuration model elements to corresponding configuration space model elements to find potential variability points for adaptations. Regarding the same dimension of abstraction, *Implementation Models* can be seen as refinements of *Configuration and Architectural Models* as they describe how a configuration and architecture is actually realized using concrete technologies. Thus, refinement relations are conceivable between models of these two categories.

Another relation can reflect the deployment or resource utilization of a system by means of relating *Architectural Model* elements and *Resource Model* elements, or in other words, which components of a running systems are deployed on which nodes and are consuming which resources. *Context and Resource Models* can also refer to *Configuration Space and Variability Models* since the configuration space

and variability of a system can be influenced by the current context or resource conditions. For example, a certain variant is disabled due to limited resources.

Besides relations between models of different categories, there can also exist relations between models of the same category. For example, in [7] several *Architectural Models* are employed reflecting the same system, but providing different views. However, these views overlap with each other, which can be considered as a kind of relation between these models. Furthermore, each model focuses on a certain non-functional concern, like performance, and any adaptation optimizing one concern might interfere with another concern. Thus, overlaps, trade-offs or conflicts between concerns respectively between the models are conceivable.

Finally, considering the levels of models, metamodels, and meta-metamodels, there exists conformance and instance-of relations between models of those levels.

The presented exemplars show that runtime models are usually not independent from each other, but they rather compose a network of models. Therefore, besides the runtime models also the relations between those models have to be managed at runtime. The concrete relations emerging in an approach depend, among others, on the purposes of the approach, the domain of the system and especially on the models that are employed.

3 Megamodels at Runtime

As it turned out in the previous sections, for runtime management different kinds of models and relations between models emerge. In such scenarios, it is important that these relations are maintained at runtime because this makes the relations explicit and, therefore, amenable for reasoning or analysis purposes. For example, an impact analysis is leveraged when knowing which models are related with each other. Then, the impact of any model change to related models can be analyzed by following transitively the relations and propagating the change. Moreover, relations can be classified, for example in critical and non-critical ones, and for certain costly analyses only the critical relations may be considered.

Nevertheless, relations to other models are usually not covered originally by all models because they were not foreseen when designing the corresponding modeling languages. Thus, a language for explicitly specifying all kinds of relations between various models is required for supporting the management of runtime models. Rather than applying ad-hoc and code-based solutions to relate models with each other, megamodels provide a language that supports the modeling of arbitrary models and relations between those models. Therefore, the management of models and relations itself is done in a model-driven manner enabling the use of existing MDE techniques for it. In general, megamodels for the model-driven development serve organizational and utilization purposes that should also be leveraged at runtime. Organizational purposes are primarily about managing the complexity of a multitude of models. Therefore, megamodels help in organizing a huge set of different models together with their relations by storing and categorizing them. According to Bézivin et al., megamodels act as some kind of registry for models [12] or even as a global map for the information assets of a company [10]. Likewise, megamodels can serve as a means to organize and

maintain runtime models and their relations in the domain of runtime system management since several models and relations can be simultaneously employed at runtime (cf. Sections 2.1 and 2.2).

Utilization purposes of megamodels are primarily about navigation and automation. Megamodels can be the basis for navigating through models by using relations between models. Thus, starting from a model, all related models can be reached in a model-driven manner instead of using mechanisms at a lower level of abstraction like programming interfaces. Having the conceivable relations between runtime models in mind (cf. Section 2.2), navigating between models at runtime is essential for a comprehensive system management approach.

Automation aims at increasing the efficiency by treating relations between models as executable units that take models as input and produce models as output. Thus, a megamodel can be considered as an executable process, and additional automations for executing a megamodel can be defined on top of a megamodel. For example, a megamodel can be used to automatically analyze the impact of model changes to other related models. Therefore, relations can be used to synchronize model changes to related models and these synchronized models are then analyzed to investigate the impact of the initial changes. This can be used at runtime to validate a planned adaptation on different models before the system is actually adapted. Finally, automation also considers the maintenance of models and relations, which should be automated as far as possible since models and relations are often both dynamic and they change over time.

Having outlined the application of megamodels at runtime, the following section presents two case studies exemplifying megamodels at runtime.

4 Case Studies

In this section, we outline two case studies from our previous works and how these case studies benefit from the application of megamodels at runtime.

4.1 IT Service Management

In [16] we presented a model-driven configuration management system (CMS) for advanced IT service management (ITSM) by applying several MDE techniques. The core of a model-driven CMS is a configuration management database (CMDB) that stores an as-is and a to-be *Configuration Model* of a managed system. Configuration models consist of configuration items and relations between items, while items are manageable units of a managed system, like servers or applications. On top of a model-driven CMS, we realized three simplified ITIL processes by using MDE techniques, namely, change management, release & deployment management, and service asset & configuration management.

The service asset & configuration management process is responsible for providing an up-to-date as-is *Configuration Model* in a CMDB. Furthermore, key performance indicators (KPIs) are implemented to provide more control on this process. An example KPI is the degree of discrepancy between the to-be and the as-is configuration models, which is the number of covered configuration items in both models divided by the number of items in the to-be configuration model.

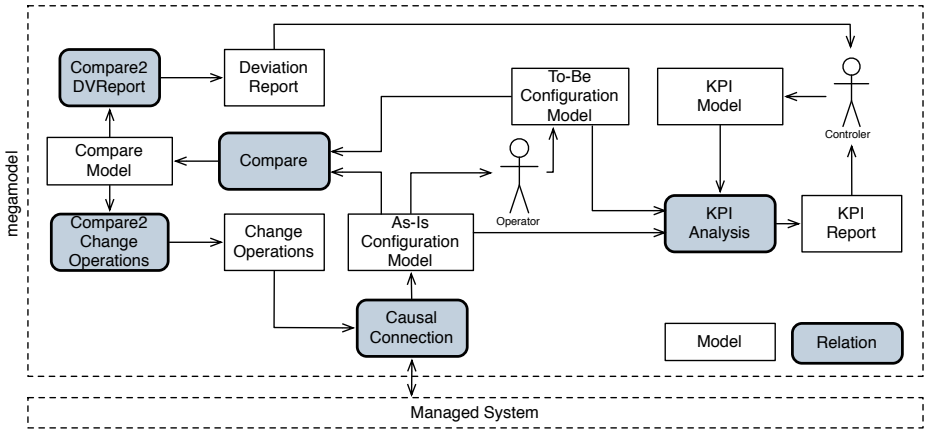

Fig. 2. A simplified megamodel example for ITSM

The change management process provides capabilities to define changes to a managed system based on models. Thus, an operator models changes directly on the as-is configuration model and then stores it as a to-be configuration model, which is further used by the release & deployment process to perform the modeled changes. Therefore, a set of change operations are automatically derived by comparing the defined to-be with the as-is configuration model.

Such a CMS can be appropriately captured by a megamodel, which is shown as a simplified example for ITSM in Figure 2. Additional actors are integrated for indicating manual interventions. The megamodel shows the models used in this system and the relations between these models. The *As-Is* and the *To-Be Configuration Models* belong to the category of *Configuration and Architectural Models*, and they are both used for the *KPI Analysis*. This analysis evaluates the KPIs specified as rules in the *KPI Model*, which therefore belongs to the model category of *Rules*, and the analysis results are described in a *KPI Report*.

Furthermore, model relations can be mapped to operations that are automatically executed, e.g., the *Compare* relation is implemented by an EMF Compare[1] operation or the *Compare2Change Operations* is a model transformation. Thus, whenever changes occur, i.e., the *To-Be Configuration Model* is modified, *Change Operations* are automatically derived and performed on the system, while the *KPI Analysis* observes the progress of performing the changes to the system.

4.2 Self-Adaptive Software

In the field of self-adaptive software, we presented an approach that employs several runtime models simultaneously for monitoring [7] and adapting [17] a system. This is outlined in Figure 3. A running *Managed System* is reflected by an *Implementation Model* and both are causally connected. However, the implementation model is platform-specific, complex, at a low level of abstraction, and related to the solution space of the system. Therefore, abstract runtime models

[1] Eclipse Modeling Framework Compare, `http://wiki.eclipse.org/EMF_Compare`

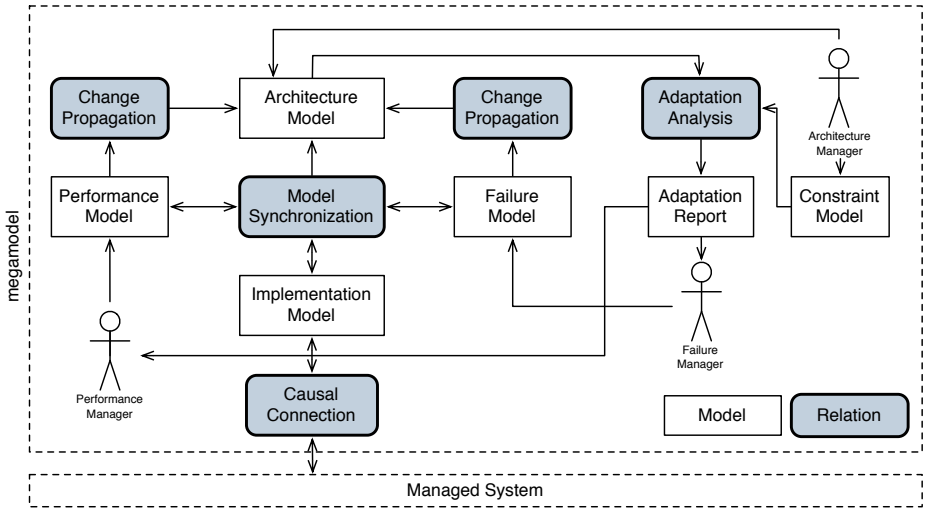

Fig. 3. A megamodel example for self-adaptive software

are derived from the implementation model using incremental and bidirectional *Model Synchronization* techniques. These abstract models can be causally connected to the system via the implementation model, and they belong to the category of *Configuration and Architectural Models*. Each of these abstract models focuses on a specific concern of interest, which leverages models related to problem spaces. An *Architecture Model*, a *Performance Model*, and a *Failure Model* are derived focusing on architectural constraints, performance, and failures of the system, respectively. Thus, specific self-management capabilities are supported by distinct models, like self-healing by the failure model or self-optimization by the performance model. Consequently, specialized autonomic managers, like a *Performance Manager* working on the performance model, can be employed.

However, adaptations performed by a certain manager due to a certain concern might interfere with other concerns covered by other managers. For example, adaptations based on the performance model, like deploying an additional component to balance the load, might violate architectural constraints covered by the *Architecture Model*, like the affected component can only be deployed once.

Since each concern is covered by a different model, megamodels can be used to describe relations, like interferences or trade-offs, between different models or concerns. Moreover, the coordination between different managers can be modeled with megamodels, which can be enacted at runtime to balance competing concerns, as outlined by the following scenario. Before any adaptation proposed by the performance or failure manager who change the performance or failure model, respectively, is executed on the system by triggering the *Model Synchronization*, the changes are automatically propagated to the architecture model (cf. *Change Propagation* relations in Figure 3). Then, the architecture manager takes the updated architecture model and the *Constraint Model* to analyze and validate the proposed adaptations (*Adaptation Analysis*). The resulting *Adap-*

tation Report is sent to the manager proposing the adaptation and it instructs either the execution of the proposed adaptation on the system or the rollback of the corresponding model changes depending on the analysis results.

Both presented case studies exemplified potential use cases for megamodels at runtime and benefits of megamodels for advanced system management approaches using multiple runtime models simultaneously.

5 Conclusion and Future Work

In this paper we have shown that the issue of complexity in the domain of model-driven development, caused by the amount of models and their relations, is also a problem in the domain of runtime system management and runtime models. Since for the latter domain this problem is rather neglected, we addressed it by presenting a categorization of runtime models and potential relations that can exist between models of the same or different categories. Based on that, we showed that megamodels are an appropriate formalism to manage runtime models and their relations. This has been exemplified by two case studies outlining the benefits in the domain of runtime system management by providing a high level of automation for organizing and utilizing runtime models and relations.

As future work, we plan to elaborate our categorization to incorporate other preliminary classifications comparing development and runtime models [1, 38] and describing dimensions of runtime models [2]. This includes possible categorizations of relations between runtime models. Finally, to evaluate this proposal, we will investigate the application of our megamodel approach designed for the development and deployment time [39] to the domain of runtime management.

References

1. France, R., Rumpe, B.: Model-driven Development of Complex Software: A Research Roadmap. In: Proc. of the ICSE Workshop on Future of Software Engineering (FOSE), IEEE (2007) 37–54
2. Blair, G., Bencomo, N., France, R.B.: Models@run.time: Guest Editors' Introduction. Computer **42**(10) (2009) 22–27
3. Song, H., Xiong, Y., Chauvel, F., Huang, G., Hu, Z., Mei, H.: Generating Synchronization Engines between Running Systems and Their Model-Based Views. In: Models in Software Engineering, Workshops and Symposia at MODELS 2009. Volume 6002 of LNCS. Springer (2010) 140–154
4. Morin, B., Barais, O., Jézéquel, J.M., Fleurey, F., Solberg, A.: Models@Run.time to Support Dynamic Adaptation. Computer **42**(10) (2009) 44–51
5. Oreizy, P., Medvidovic, N., Taylor, R.N.: Architecture-based runtime software evolution. In: Proc. of the 20th Intl. Conference on Software Engineering (ICSE), IEEE (1998) 177–186
6. Garlan, D., Cheng, S.W., Huang, A.C., Schmerl, B., Steenkiste, P.: Rainbow: Architecture-Based Self-Adaptation with Reusable Infrastructure. Computer **37**(10) (2004) 46–54
7. Vogel, T., Neumann, S., Hildebrandt, S., Giese, H., Becker, B.: Incremental Model Synchronization for Efficient Run-Time Monitoring. In: Models in Software Engineering, Workshops and Symposia at MODELS 2009. Volume 6002 of LNCS. Springer (2010) 124–139
8. Bencomo, N., Blair, G., France, R., Munoz, F., Jeanneret, C.: 4th International Workshop on Models@run.time. In: Models in Software Engineering, Workshops and Symposia at MODELS 2009. Volume 6002 of LNCS. Springer (2010) 119–123
9. Mellor, S.J., Scott, K., Uhl, A., Weise, D.: MDA Distilled: Principles of Model-Driven Architecture. Addison-Wesley, Boston (2004)
10. Bézivin, J., Gérard, S., Muller, P.A., Rioux, L.: MDA components: Challenges and Opportunities. In: 1st Intl. Workshop on Metamodelling for MDA. (2003) 23–41
11. Favre, J.M.: Foundations of Model (Driven) (Reverse) Engineering : Models – Episode I: Stories of The Fidus Papyrus and of The Solarus. In: Language Engineering for Model-Driven Software Development. Number 04101 in Dagstuhl Seminar Proceedings, IBFI, Schloss Dagstuhl (2005)
12. Bézivin, J., Jouault, F., Valduriez, P.: On the Need for Megamodels. In: Proc. of the OOPSLA/GPCE Workshop on Best Practices for Model-Driven Software Development. (2004)

13. Barbero, M., Fabro, M.D.D., Bézivin, J.: Traceability and Provenance Issues in Global Model Management. In: ECMDA-TW'07: Proc. of 3rd Workshop on Traceability. (2007) 47–55
14. Workshop on Models@run.time, http://www.comp.lancs.ac.uk/~bencomo/MRT/ (2006-2009)
15. Vogel, T., Neumann, S., Hildebrandt, S., Giese, H., Becker, B.: Model-Driven Architectural Monitoring and Adaptation for Autonomic Systems. In: Proc. of the 6th Intl. Conference on Autonomic Computing and Communications (ICAC), ACM (2009) 67–68
16. Giese, H., Seibel, A., Vogel, T.: A Model-Driven Configuration Management System for Advanced IT Service Management. In: Proc. of the 4th Intl. Workshop on Models@run.time. Volume 509 of CEUR-WS.org. (2009) 61–70
17. Vogel, T., Giese, H.: Adaptation and Abstract Runtime Models. In: Proc. of the 5th ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), ACM (2010) 39–48
18. Jouault, F., Bézivin, J., Chevrel, R., Gray, J.: Experiments in Run-Time Model Extraction. In: Proc. of 1st Intl. Workshop on Models@run.time. (2006)
19. Kuhn, A., Verwaest, T.: FAME - A Polyglot Library for Metamodeling at Runtime. In: Proc. of the 3rd Intl. Workshop on Models@run.time, Technical Report COMP-005-2008, Lancaster University (2008) 57–66
20. Costa, F., Provensi, L., Vaz, F.: Towards a More Effective Coupling of Reflection and Runtime Metamodels for Middleware. In: Proc. of 1st Intl. Workshop on Models@run.time. (2006)
21. Gjerlufsen, T., Ingstrup, M., Wolff, J., Olsen, O.: Mirrors of Meaning: Supporting Inspectable Runtime Models. Computer **42**(10) (2009) 61–68
22. Maoz, S.: Using Model-Based Traces as Runtime Models. Computer **42**(10) (2009) 28–36
23. Goldsby, H.J., Cheng, B.H., Zhang, J.: AMOEBA-RT: Run-Time Verification of Adaptive Software. In: Models in Software Engineering: Workshops and Symposia at MoDELS 2007. Volume 5002 of LNCS. Springer (2008) 212–224
24. Maoz, S.: Model-Based Traces. In: Proc. of the 3rd Intl. Workshop on Models@run.time, Technical Report COMP-005-2008, Lancaster University (2008) 16–25
25. Höfig, E., Deussen, P.H., Coskun, H.: Statechart Interpretation on Resource Constrained Platforms: a Performance Analysis. In: Proc. of the 4th Intl. Workshop on Models@run.time. Volume 509 of CEUR-WS.org. (2009) 99–108
26. Sanchez, M., Barrero, I., Villalobos, J., Deridder, D.: An Execution Platform for Extensible Runtime Models. In: Proc. of the 3rd Intl. Workshop on Models@run.time, Technical Report COMP-005-2008, Lancaster University (2008) 107–116
27. Dey, A.K.: Understanding and Using Context. Personal Ubiquitous Comput. **5**(1) (2001) 4–7
28. Schneider, D., Becker, M.: Runtime Models for Self-Adaptation in the Ambient Assisted Living Domain. In: Proc. of the 3rd Intl. Workshop on Models@run.time, Technical Report COMP-005-2008, Lancaster University (2008) 47–56
29. Acher, M., Collet, P., Fleurey, F., Lahire, P., Moisan, S., Rigault, J.P.: Modeling Context and Dynamic Adaptations with Feature Models. In: Proc. of the 4th Intl. Workshop on Models@run.time. Volume 509 of CEUR-WS.org. (2009) 89–98
30. Cetina, C., Giner, P., Fons, J., Pelechano, V.: Autonomic Computing through Reuse of Variability Models at Runtime: The Case of Smart Homes. Computer **42**(10) (2009) 37–43
31. Elkhodary, A., Malek, S., Esfahani, N.: On the Role of Features in Analyzing the Architecture of Self-Adaptive Software Systems. In: Proc. of the 4th Intl. Workshop on Models@run.time. Volume 509 of CEUR-WS.org. (2009) 41–50
32. Ferry, N., Hourdin, V., Lavirotte, S., Rey, G., Tigli, J.Y., Riveill, M.: Models at Runtime: Service for Device Composition and Adaptation. In: Proc. of the 4th Intl. Workshop on Models@run.time. Volume 509 of CEUR-WS.org. (2009) 51–60
33. Bencomo, N., Whittle, J., Sawyer, P., Finkelstein, A., Letier, E.: Requirements reflection: requirements as runtime entities. In: Proc. of the 32nd ACM/IEEE Intl. Conference on Software Engineering (ICSE), ACM (2010) 199–202
34. Fleurey, F., Solberg, A.: A Domain Specific Modeling Language Supporting Specification, Simulation and Execution of Dynamic Adaptive Systems. In: Proc. of the 12th Intl. Conference on Model Driven Engineering Languages and Systems (MODELS). Volume 5795 of LNCS., Springer (2009) 606–621
35. Ramirez, A.J., Cheng, B.H.: Evolving Models at Run Time to Address Functional and Non-Functional Adaptation Requirements. In: Proc. of the 4th Intl. Workshop on Models@run.time. Volume 509 of CEUR-WS.org. (2009) 31–40
36. Dubus, J., Merle, P.: Applying OMG D&C Specification and ECA Rules for Autonomous Distributed Component-based Systems. In: Proc. of 1st Intl. Workshop on Models@run.time. (2006)
37. Cheng, B.H., Sawyer, P., Bencomo, N., Whittle, J.: A Goal-Based Modeling Approach to Develop Requirements of an Adaptive System with Environmental Uncertainty. In: Proc. of the 12th Intl. Conference on Model Driven Engineering Languages and Systems (MODELS). Volume 5795 of LNCS., Springer (2009) 468–483
38. Bencomo, N.: On the Use of Software Models during Software Execution. In: Proc. of the ICSE Workshop on Modeling in Software Engineering (MISE), IEEE (2009) 62–67
39. Seibel, A., Neumann, S., Giese, H.: Dynamic Hierarchical Mega Models: Comprehensive Traceability and its Efficient Maintenance. Software and Systems Modeling **9** (2009) 493–528