Softw Syst Model (2016) 15:31–69
DOI 10.1007/s10270-013-0394-9

REGULAR PAPER

Systematic literature review of the objectives, techniques, kinds, and architectures of models at runtime

Michael Szvetits · Uwe Zdun

Received: 10 April 2013 / Revised: 24 November 2013 / Accepted: 29 November 2013 / Published online: 17 December 2013

Abstract In the context of software development, models provide an abstract representation of a software system or a part of it. In the software development process, they are primarily used for documentation and communication purposes in analysis, design, and implementation activities. Model-Driven Engineering (MDE) further increases the importance of models, as in MDE models are not only used for documentation and communication, but as central artefacts of the software development process. Various recent research approaches take the idea of using models as central artefacts one step further by using models at runtime to cope with dynamic aspects of ever-changing software and its environment. In this article, we analyze the usage of models at runtime in the existing research literature using the Systematic Literature Review (SLR) research method. The main goals of our SLR are building a common classification and surveying the existing approaches in terms of objectives, techniques, architectures, and kinds of models used in these approaches. The contribution of this article is to provide an overview and classification of current research approaches using models at runtime and to identify research areas not covered by models at runtime so far.

Keywords Models · Runtime · Literature review

Communicated by Prof. Gordon Blair.

M. Szvetits (✉)
Software Engineering Group, Information Technology Institute,
University of Applied Sciences Wiener Neustadt,
Wiener Neustadt, Austria
e-mail: michael.szvetits@fhwn.ac.at

U. Zdun
Software Architecture Group, Faculty of Computer Science,
University of Vienna, Vienna, Austria
e-mail: uwe.zdun@univie.ac.at

1 Introduction

In traditional software engineering, the development process is divided into several activities or phases reaching from requirements specification over software construction to deployment and maintenance. These software development phases and software execution are strictly separated from each other, and modification of running software is done by re-deployment of the changed software components, e.g. in binary or bytecode form. However, in recent years, the distinction of software development and execution blurs, for example because modern applications have to adapt themselves according to changing requirements and execution environments while they are up and running [102]. A key idea to overcome limitations in flexibility is to reuse model artefacts from the software development phases at runtime [20,34]. Information of the running system is fed back to the software models to support reasoning at runtime and to take corrective actions at the model level. Furthermore, performing changes at the model level of those models at runtime improves the synchronization between design artefacts and the implementation of the software system [107]. To realize such a connection between a running system and its models, the running system needs a self-representation of itself based on models which are causally connected to it [34,166].

Based on the mentioned utilization of models, a runtime model is defined as *abstraction of a running system that is being manipulated at runtime for a specific purpose* [27]. An alternative definition is given by Blair et al. [34] as *causally connected self-representation of the associated system that emphasizes the structure, behaviour or goals of the system from a problem space perspective*.

Different challenges might arise when using software models at runtime to realize dynamic software system behaviour. For instance, one might be required to choose an

adequate adaptation strategy, apply introspection mechanisms to running systems, check the formal correctness of models, compare models to calculate adaptation operations, or transform models to simplify reasoning, to name just a few issues that must be addressed. Since there is a wide range of different software models and many application scenarios for models at runtime, a need for classification in terms of objectives, techniques, architectures and kinds of models used at runtime arises.

In this article, we inspect the existing research literature which tackles the topic of models at runtime and extract information regarding the aforementioned classification. We accomplish this by performing a Systematic Literature Review (SLR) [150,151] consisting of an initial search phase, multiple filter phases with well-defined selection criteria and a final classification phase. For each classification, we present relevant approaches found in our literature review to provide a self-contained overview of the topic. Regarding selection criteria, we categorized the approaches into included, non-detailed, and excluded literature, where approaches of the non-detailed category are presented only in short form for the sake of completeness. The result of our SLR is an overview of current research approaches using models at runtime, as well as a common classification scheme focussing on the objectives, techniques, architectures and kinds of models used in these approaches. We also categorize the actions after runtime model analysis documented in the literature, as an important aspect of runtime model architectures (i.e. how runtime models are connected to the running system). Finally, we summarize the existing empirical evidence provided for models at runtime in the literature and identify research areas not covered in detail by models at runtime so far.

This article is structured as follows: In the next section, we will describe our research method to identify the relevant literature using models at runtime. In Sect. 3, we present our research results by explaining the identified classification and describing relevant approaches according to this classification. In addition, we discuss possible actions after model analysis as well as empirical evidence of our identified runtime model approaches. In Sect. 4, we discuss our literature research method and results in terms of possible inaccuracies, precision of the results, possible distortions by selection criteria, perspectives, and limitations of this study. Finally, we conclude in Sect. 5.

2 Research method

The Systematic Literature Review (SLR) method is a well-defined and rigorous method of identifying, evaluating, and interpreting all available research relevant to a particular research question, topic area, or phenomenon of interest [150]. SLRs aim to present a fair, unbiased, and credible evaluation of a particular research topic [150]. This research has been conducted as an SLR by following the guidelines of Kitchenham et al. [150,151], who propose three main phases: Planning the review (identify the main rationale for undertaking the review and develop a review protocol), executing the review (conduct the review by executing the planned review protocol from the previous phase), and reporting the review (present the results of the review and its dissemination to the interested parties). Phases involved in the SLR appear to be sequential but they are usually iterative, e.g. search terms, inclusion criteria, and exclusion criteria can be refined while the review is in progress. In this section, the review plan (first phase) and the review execution (second phase) of our SLR are described. The outcomes of the third phase, the reporting phase, are presented in Sect. 3, where we report our research results, and in Sect. 4, where we discuss these results.

We first introduce our general search strategy and then the research questions to be answered by our literature analysis. Next we define our literature selection criteria to extract relevant publications regarding models at runtime. We then give a detailed insight into our literature selection process. The key idea of finding relevant literature is to perform an initial search, filter the results, check references of selected publications, and finish the search by grouping and duplicate elimination.

2.1 Search strategy

Our SLR is based on an electronic search in two publication databases: The Association for Computing Machinery (ACM) Guide to Computing Literature (accessed through the ACM Digital Library[1]) and the IEEE Xplore Digital Library.[2] It aims to provide a comprehensive coverage of bibliographic citations from all major publishers in the field of computing. At the time of this SLR, the ACM Guide to Computing Literature contained 2,150,450 and the IEEE Xplore Digital Library 3,511,057 bibliographic records. The methodology, which is exploited for comprehensive coverage, is to perform an advanced search in these databases for accumulating bibliographic citations from the major publishers in computing. The search was based on the titles and abstracts of the articles. The rationale behind using this particular method is to find the biggest share of scientific citations that are relevant to answer the specific research questions listed below.

This search strategy might inevitably miss some useful citations, for instance, because some minor publishers are not included in these databases and some (especially) older publications have a very short or not very meaningful abstract. In response to this problem, so-called "snowballing" [42] (i.e.

[1] See: http://dl.acm.org/.

[2] See: http://ieeexplore.ieee.org/.

Table 1 Overview of the applied literature selection criteria

Inclusion criteria (covered in detail)
• Articles found in the ACM Guide to Computing Literature or IEEE Xplore Digital Library
• Articles found through snowballing till August 2013, the time this study was conducted
• Articles published in peer-reviewed journals, conferences, and workshops
Inclusion criteria (covered in less detail)
• Doctoral dissertations
• Publications that use models at runtime but do not discuss the runtime aspects of them (e.g. execution of a static state machine model)
• Publications that contain models not related to software development (e.g. mathematical models, thermal models of hardware)
• Summary or survey publications because of their reduced level of detail (we discuss those in Sect. 3.9, if appropriate)
• Publications focussing on user interface generation/adaptation, as they are too specific
Exclusion criteria
• Books, web sites, technical reports, and master theses
• Publications without abstracts
• Publications in which the usage of runtime models is not clearly understandable (e.g. they have a different focus and the model usage is covered in insufficient detail)

following the references of articles found during the search) is performed in an iterative way to identify additional citations. That is, the bibliographies of every selected publications are checked for useful articles that are missed in the initial search. Snowballing is performed until a convergence is reached and no more new relevant articles are found.

2.2 Research questions

By performing a systematic literature review consisting of multiple phases, we are looking for answers to the following research questions:

- For what purposes are models used at runtime?
- Which techniques and kinds of models are used when processing models at runtime?
- Which problems are addressed by using models at runtime?
- Which architectures exist for processing models at runtime?
- Which research methods are most frequently used for evaluating runtime model approaches? Which empirical evidence has been reported?

Answers to these research questions help us to provide an overview of current research approaches and to identify research areas not covered in detail by models at runtime so far. Furthermore, answering the aforementioned questions enables us to build a common classification for objectives, techniques, architectures and kinds of models used at runtime. We introduce our resulting classification in Sect. 3 when presenting our research results.

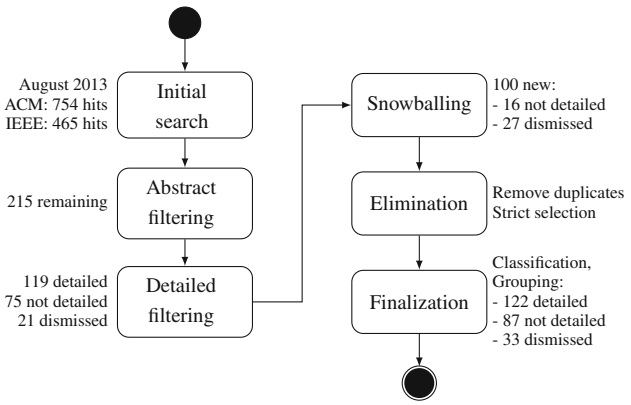

Fig. 1 The literature search process

2.3 Literature selection criteria

To conclude efficiently about our research questions, we defined selection criteria which help to decide whether a publication will be excluded, included (covered in detail), or covered in less detail when applying our search process. Table 1 gives an overview of these selection criteria. Regarding snowballing mentioned in the table, we included related work from all publications which are covered in detail.

2.4 Literature search process

The literature search process is divided into six steps: An initial search phase, two filtering phases, a snowballing phase, an elimination phase, and a finalization phase. The process is depicted in Fig. 1 and shows all interim results between the phases.

In the initial search phase, an appropriate search string had to be found, capturing all relevant publications regarding models at runtime. Our search string is based on the Models@run.time workshop proceedings [24–29], the publications presented in the Dagstuhl Seminar 11481 [13] and the publications in special issues concerned with models at runtime (the last from March 2013 [44]). Our goal was that every relevant paper presented in this workshop would be covered by our search. As a result, the following search terms either in abstract or title of a publication led to an immediate inclusion in the search results:

$$P = \{models\ at\ runtime, models\ for\ runtime, runtime\ models\}$$

Alternatively, a conjunction of the following terms in the abstract led also to inclusion:

$$Q = \{model, meta\text{-}model\}$$
$$R = \{runtime\}$$
$$S = \{dsl, model\text{-}driven, meta\text{-}model\text{-}driven, model\text{-}based, dsml, system\ at\ runtime, application\ model, abstract\ model, engineering\ model, architectural\ model, feature\ model, behaviour\ model, model\ of\ system, inferred\ model, dynamic\ model, runtime\ state\}$$

Hence, the overall search string can be combined in the following way:

$$Result = P \vee (Q \wedge R \wedge S)$$

We were aware of multiple spellings of words and abbreviations when applying our search string (e.g. runtime, runtime, run time). This led to a search string with a total length of 1495 characters. The initial search ended on 7 August 2013 and yielded 754 publications in the ACM Digital Library and 465 publication in the IEEE Xplore Digital Library.

The second phase addressed the elimination of duplicates introduced by the two publication databases and the filtering of the publications by applying the aforementioned selection criteria on their abstracts (as far as possible). In case of uncertainty regarding inclusion/exclusion, articles were included for further analysis in subsequent phases. In this phase, a first classification was created as a basis for further refinement throughout the literature search process. The filtering yielded 215 remaining publications.

In the next phase, a more detailed filtering was applied by inspecting the whole content of the remaining 215 publications, applying the literature selection criteria and classifying them into three groups: 119 *detailed* (clearly fitting the topic), 75 *not detailed* (fitting the topic, but insufficiently detailed or too specific), and 21 *dismissed* (not fitting the topic). Furthermore, the initial paper classification of the previous phase was updated for further refinement.

The fourth phase addressed the analysis of the references of the publications inspected in detail. This snowballing phase added 100 new papers to our search results, out of which 16 where categorized as *not detailed* and 27 as *dismissed*.

The first step of the next phase was to remove duplicates introduced by the snowballing phase. In the second step, we revised our aforementioned group clustering by strictly applying the selection criteria to papers included under uncertainty before.

In the last phase, we organized our overall search results by grouping same approaches which were published multiple times, leading to distinct groups consisting of 122 detailed, 87 non-detailed, and 33 dismissed publications. Studies having multiple published descriptions (collected in one distinct group) are included only once by using the most detailed and up-to-date version of the study. After applying elimination and finalization, only 15 out of the 100 papers added by snowballing remained in the results, giving us confidence that our search string captured the majority of the relevant literature.

3 Research results

In this section, we outline the problems tackled by models at runtime as well as the identified objectives, techniques, kinds of models, and architectures used in combination with runtime models. Furthermore, we discuss summary and survey publications identified by our literature research in Sect. 3.9.

3.1 Problems

In our SLR, we identified several problems to be resolved by the usage of models at runtime. Figure 2 gives an overview of these problems by grouping them into different categories. The categories are described top-down according to the picture in the text below. In the discussion of the approaches in the subsections below, we explain for the individual approaches which of these problems they address.

Inaccurate predictions result from unknown requirements evolution [30,215,227,274] and associated change impacts as well as difficult performance predictions. In a changing environment, it is no easy task to monitor and visualize the impact of adaptation rules which are applied when a system should meet new requirements. Models at runtime help to visualize such change impacts by analyzing affected software parts at the model level and checking whether specified application constraints have been violated [261]. Furthermore, interconnected runtime models can be traversed to highlight influenced models and their elements if one model changes at runtime [229]. *Changing and heterogeneous environments* also complicate predictions, as they add some level

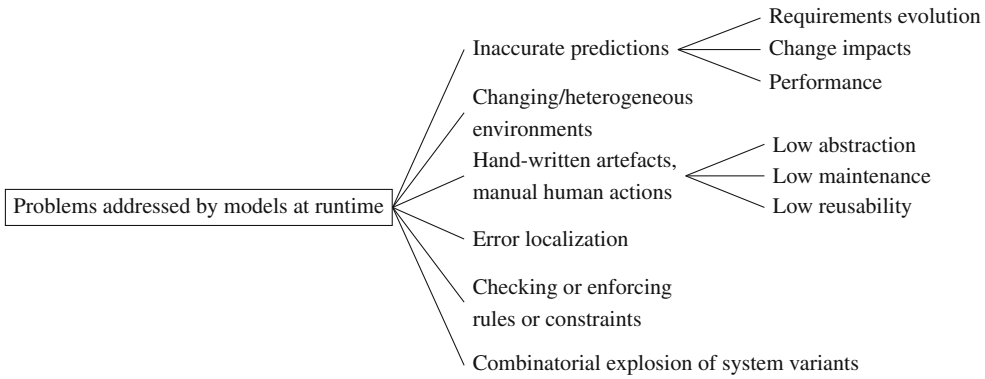

Fig. 2 Problems addressed by models at runtime

of uncertainty regarding stability and satisfaction of context-related constraints. In addition, to cope with these changing and heterogeneous environments, often flexible adaptation is needed by system engineers [117], but many approaches mix adaptation logic with application behaviour by using conditional expressions and other low-level programming features, leading to bad scalability. Models help to increase such needed flexibility in adaptation to tackle problems of inaccurate predictions and changing environments, e.g. by using queueing models and Markov models to predict timing and future states [47,51,81,104,184], Dynamic Decision Networks for enhanced decision-making in self-adaptive systems [21,22] or architecture models to analyze system-wide change impacts [101]. Increased adaptation is one main objective of models at runtime and is discussed in Sect. 3.3.1.

Creating *hand-written artefacts* or performing other *manual human actions* is often time-consuming as well as error-prone and leads to low abstraction, maintenance, and reusability. A common solution to solve these problems is to use MDE techniques to auto-generate software artefacts and reconfiguration scripts responsible for switching between runtime configurations [87,182]. Architecture models are an example of information sources when calculating reconfiguration actions, especially in combination with model comparison techniques to compare a target architecture model with the model of the current system configuration. Model comparison and other model processing techniques are discussed in Sect. 3.4.

As for *error localization*, it is hard to predict where errors will occur in highly dynamic environments. Runtime models provide system operators with enhanced capabilities to localize faults in behavioural models like workflows [195]. Beyond that, with runtime models faults can not only be localized, but also eliminated by architecture-based self-repair [52].

A problem commonly addressed by models at runtime is *checking or enforcing rules or constraints*. That is, the model contains or implies rules or constraints, such as consistency requirements with the running system or other artefacts or business policies that should not get violated. Techniques operating on runtime models are used to check (monitor) or even enforce these rules or constraints at runtime.

Another problem addressed by models at runtime is the *combinatorial explosion* of possible system configurations when calculating the most rewarding adaptation plan [178–181], which can further be complicated when checking a great number of model elements at runtime because of increased execution time and memory occupation [81]. Combinatorial explosion can be tackled by using aspect-oriented programming to model system variation points instead of concrete configurations [178–181]. Feature models describe a hierarchical composition of system parts and declare them either as mandatory or optional, simplifying adaptation at runtime by reducing the set of possible system part combinations. Filieri et al. [81] propose a pre-computation step regarding Discrete Time Markov Chains to tackle the problem of state explosion which occurs in analyzing the model. They use transition variables that are the parameters of the model whose value becomes known only at runtime. The output produced by the pre-computation step is a set of symbolic expressions which can easily be evaluated at runtime by replacing the variables with the real values gathered by monitoring the system, thus shifting the cost of model analysis to design time. Adaptation scenarios and strategies are discussed in detail in Sect. 3.3.1.

3.2 Classification overview

Early approaches using models at runtime focussed on architecture-based mechanisms to realize monitoring and adaptation [95,96,98]. The ideas further evolved from early middleware-based adaptation approaches [63,65,74,152,210] to other topics like error handling [135,145] and model execution (see Sect. 3.4.6). Recent approaches focus on goal- and requirement-oriented aspects [53,215,260]. There has been a recognizable shift from simple adaptation interests to wider application fields and more goal-oriented and user-centric approaches over the last years [30] (see Sect. 4.4).

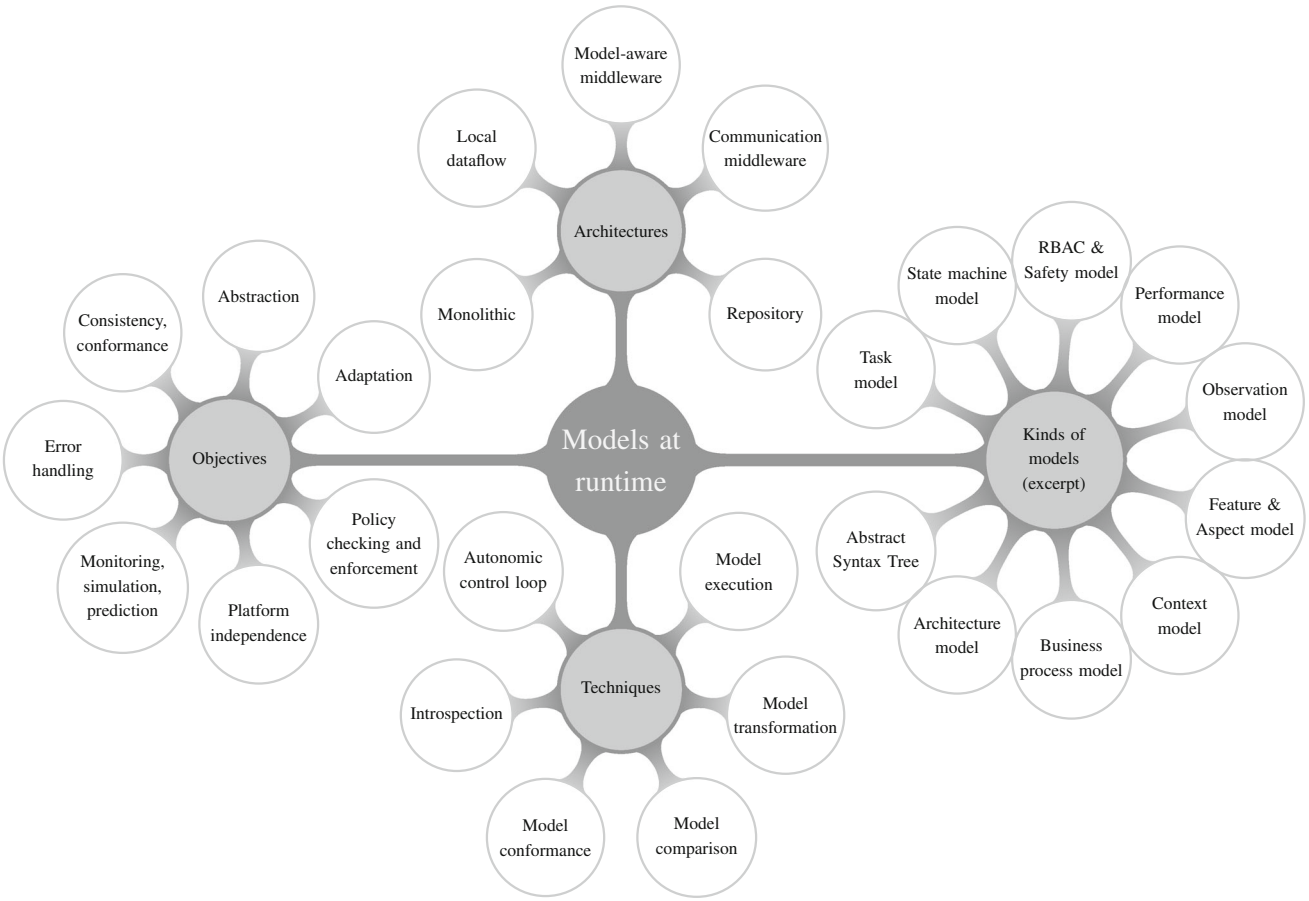

Fig. 3 Overview of objectives, techniques, kinds and architectures when using models at runtime

This shift leads to a great variety of objectives (e.g. adaptation, monitoring, error handling) which are achieved by using different kinds of models at runtime (e.g. architectural models, goal models). Architectures and techniques must be selected accordingly to support the realization of the objectives with their corresponding kinds of models.

Since we not only want to identify research areas not covered so far, but also provide a comprehensive overview for researchers who are new to this topic, we decided to analyze the mentioned aspects (objectives, techniques, kinds of models, architectures) because they bring answers to fundamental questions:

- What do we need models at runtime for? *(objectives)*
- Which models can be used at runtime? *(kinds of models)*
- How can we realize systems using models at runtime? *(techniques)*
- Where must models reside to support the running system? *(architectures)*

A discussion about the interconnections of these aspects is given in Sect. 4.2. Figure 3 gives an overview of the classification according to these aspects. Each shown aspect (architectures, kinds, techniques, objectives) is further divided into several parts to refine the description of the associated aspect. These parts are covered in detail in the subsequent subsections. That is, the figure serves as an overview of our research results derived from our search phase described in Sect. 2.4.

3.3 Objectives

In this section, we observe the objectives pursued when using a system which utilizes models at runtime. We extracted the objectives from approaches identified by our literature review by collecting the objectives stated by the authors and grouping similar objectives into classes. This process resulted in seven classes of objectives addressed by runtime model (see Fig. 3):

- *Adaptation*: Change the system according to changing environment and requirements.
- *Monitoring, simulation, prediction*:
  - Monitor the system by using models which help to trace application behaviour.

  - Simulate changes at the model level to analyze their consequences.
  - Predict system properties like performance by analysis at the model level.

- *Abstraction*: Interact with the system by using models which are closer to the problem space.
- *Platform independence*: Provide platform-independent views on the system under observation.
- *Consistency and conformance*:
  - Avoid contradictions between different software parts and/or (development) artefacts.
  - Ensure conformance to other models or integrity constraints in general.

- *Policy checking and enforcement:* Adhere to policies like (real-)time and security regulations.
- *Error handling:* Enable model-based debugging, fault localization, tracing, and self-healing.

In the following sections, we will discuss each of the classes in detail and conclude with an overview of approaches related to each class of objectives (we put abstraction and platform independence into a single section since they are closely related).

3.3.1 Adaptation

One main objective of using runtime models is to ease adaptation in environments with continuously changing requirements. When adapting a system, one challenge lies in proving the correctness of the adaptation within a composed system. Inverardi et al. [144] propose a theoretical assume-guarantee framework which analyzes whether certain invariant properties are violated. Their framework can be applied at different levels of abstraction spanning from code to software architecture. For the development process itself, Inverardi and Mori [140,143] propose a software development process to support consistent evolutions with the help of context models, features models, and a control loop (see Sects. 3.5 and 3.4.1, respectively). Goldsby et al. [113] propose AMOEBA-RT, a model checker for adaptive systems which collects runtime state information via aspect-oriented techniques and checks for violations of formal specifications. Like many other approaches, AMOEBA-RT uses an automaton-based approach to determine whether the runtime information satisfies certain properties specified in temporal logic (model conformance is discussed in Sect. 3.4.3).

In our literature search, we spotted five major scenarios addressed by adaptation through runtime models: User interface adaptation, requirement changes, contextual changes, QoS enforcement, and change impact analysis. These scenarios and appropriate strategies to realize system adaptation

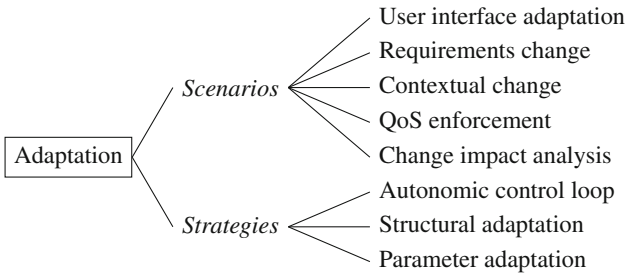

Fig. 4 Adaptation scenarios and strategies

are depicted in Fig. 4. The scenarios are described in subsequent paragraphs, while the strategies relate to techniques are discussed below in Sect. 3.4.1.

User interface adaptation User interface adaptation addresses dynamic user interface interaction, layouting, and management of multiple system perspectives. Models at runtime help to realize such scenarios by describing possible interaction flows, layout constraints, and system parts to be personalized by the user. Garzon and Cebulla [99] use a combination of three models to support automatic adaptation by learning from user interactions: A system model (describes internal behaviour), an interaction model (describes human-machine-interaction), and a personalization model (links system and interaction model and sets constraints for personalization). Other approaches use user interface adaptation for generation of multimodal interaction [14,78,80,194,245], layouting/scaling/sizing [32,79,254], multiple system perspectives [237], user interface synchronization [35], self-explanation of user interfaces [89,93] and contextual user interface adaptation [36,66,94,222,236,270]. As mentioned earlier, for these approaches we will not go into detail.

Requirements and contextual changes Requirements and contextual changes address needed modifications to meet requirement evolution and changes in operational environments. Main challenges in triggering adaptation lie in finding out alternative system configurations and analysis of found configurations regarding benefits compared to the current system state. Models at runtime help to decrease the amount of possible system configurations by modelling variable parts of the system at runtime [140,143]. An example for such models are feature models which describe a hierarchical composition of system parts and declare them either as mandatory or optional, simplifying adaptation at runtime by reducing the set of possible system part combinations. For the adaptation process itself, the strategies depicted in Fig. 4 are used to drive the transition between system configurations. Section 3.4.1 gives detail descriptions of these strategies.

A requirements-aware system needs a formal representation of the requirements to process them at runtime. The

requirements language RELAX [1,267] is a declarative requirements language for self-adaptive systems which supports the explicit expression of environmental uncertainty in requirements. RELAX is based on fuzzy branching temporal logic and provides modal, temporal and ordinal operators to express uncertainty. Example operators are SHALL to define functionality the system must always provide (invariants) and MAY/OR to define alternatives.

To manage runtime adaptation, Georgas et al. [101] build a graph of a system that captures historical configurations and corresponding behaviours of a system. The adaptive process is augmented with meta-data such as frequency of entered configurations and how long a system remained in particular configurations. An architecture runtime model supports the system and is responsible for the initial instantiation of the system and its future evolution. The historical graph consists of system configurations (nodes) and transitions (edges) between these configurations, whereas the transitions store architectural differences between configurations in a bidirectional way, thus enabling rollback and rollforward operations. A similar rollback mechanism is proposed by Barbier [16].

To meet requirements and contextual change scenarios, Floch et al. [83] use a middleware-based approach in combination with architecture runtime models to control the adaptation mechanism. The middleware transforms architecture models into runtime representations and finds all alternative components that can be plugged into component frameworks, building a set of possible system variants. The utility of each variant is calculated and reconfiguration is triggered if a variant shows higher profit than the current configuration. This enables the application to adapt itself to changing user needs and operating environments, demonstrated by the authors with mobile applications. A similar approach is used by Taconet et al. [246] which uses new context data collectors dynamically. Context-awareness models are used and updated at runtime to fulfil new application requirements. MDE is used for guiding context-aware designers in the specification of the monitoring of distributed context sources.

Elkorobarrutia et al. [76] use state machine models as a basis for component reconfiguration to enable self-healing mechanisms. In their approach, state machine models serve as interfaces between a state machine interpreter and the component (seen as blackbox), enabling manipulation of the model at runtime to alter component behaviour. McQuiggan and Lester [174] realize intelligent tutoring systems by using dynamic self-efficacy models containing data from a special feedback process. Nguyen and Colman [191] use feature models at runtime to enable web service customization for customers, leading to a manageable number of system configurations and flexibility in requirement changes.

Inverardi and Mori [141,142] propose a framework to augment a system with new requirements arising at runtime. A requirement evolution is declared inconsistent whenever new requirements do not interact correctly with currently deployed ones. At design time, a context model is defined which describes the environment that is beyond the control of the system but may influence future system evolutions. An automatic process updates the model at runtime to reflect the current environmental situation. When a new requirement is specified, a new system variant is compiled. By using reflection, the new variant is loaded and the entry method invoked to put the new variant in action.

As pointed out in Sect. 3.2 and discussed in Sect. 4.4, recent research approaches use high-level goal- and requirements models to drive the adaptation of the system. Section 4 gives an overview of approaches using these kinds of models to cope with requirements changes during runtime.

Quality-of-Service enforcement In the scenario of Quality-of-Service (QoS) enforcement, the satisfaction of non-functional properties is addressed, e.g. a certain level of performance or reliability. This is especially required when dealing with volatile operating environments and concrete timing constraints. By using models at runtime, possible future states of the system can be predicted when environment changes occur, and such states serve as a basis to analyze whether required conditions can be met in case of environmental change. QoS-related models often rely on stochastic calculations and outline data flows between system components and dynamic system behaviour to enable simulation and point out possible bottlenecks. Examples of such models are queueing models (predict timing properties), Markov models (analyze probability of future states) and runtime representations of architecture models. Figure 5 gives an excerpt from non-functional requirements which are satisfied through system adaptation. The satisfaction of these non-functional requirements is discussed in the following paragraphs.

In case of the non-functional property *performance*, Caporuscio et al. [47], Ardagna et al. [12], and Ghezzi and Tamburelli [104] use Queueing Networks as models to manage performance satisfaction. In the approach of Ardagna et al. [12], a monitor collects data of the executing system and checks the model whether non-functional requirements are violated. Based on the collected results, the approach sup-

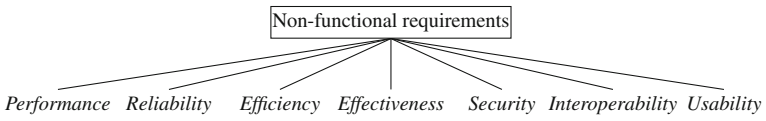

Fig. 5 Excerpt from non-functional requirements that are tackled by models at runtime

Table 2 Overview of approaches related to the classes of identified objectives

Identified objective	Related approaches
Adaptation	*User interface adaptation*: [14,32,35,36,66,67,78–80,89,93,94,99,161,194,209,222,230,236,237,245,254,270]
	Requirements and contextual changes: [16,76,83,141,142,174,191,246]
	Quality-of-Service (QoS) enforcement: [5,6,12,45,47–49,61,104,137,175,185,283]
	Change impact analysis: [203,229,261]
Monitoring, simulation, prediction	[19,31,62,77,98,103,105–108,114,115,136,160,168,169,171,177,184,198,199,207,217,231–235,255,257,258,278]
Abstraction and platform independence	[105–108,139,167,248,250,255,257,258]
Consistency/conformance checking and enforcement	[11,14,37,40,79,92,131,156,211]
Policy checking and enforcement	[15,18,123,125,132–134,138,176,206,219,220,242,272,279–281]
Error handling	[52,90,96,121,122,124,135,153,187,195,196,205,228,243,265, 271,273,275]

ports reasoning on adaptation strategies. To ensure *reliability*, Cardellini et al. [49] meet QoS requirements in volatile operating environments by using a behavioural model of the system updated at runtime by a monitoring activity. This model is used to calculate a re-arrangement of available services to fulfil performance and reliability requirements. Parameters for such a re-arrangement are derived from Service-Level-Agreements (SLAs) negotiated with service clients and providers. Calinescu et al. [45] propose the framework QoSMOS which utilizes Markov models within a feedback loop (see Sect. 3.4.1) to determine quantitatively the reliability and performance quality metrics of service-based systems. The monitoring part of the loop determines performance, reliability and workload of services and updates Markov models accordingly. The results are analyzed with the help of specified QoS requirements and changes are planned in terms of changing the implemented workflow or modifying resource allocation. These changes are then executed by modifying parameters, starting, stopping, or migrating virtual machines or using dedicated resource management mechanisms.

For *efficiency* and effectiveness, an example is the MADAM middleware which automates adaptation decision-making in reflective component-based systems to maximize the utility of an application, e.g. through minimum resource consumption, maximum resource consumption (likely to be best performance) or maximum efficiency (ratio utility to resource consumption) [5,6]. The approach is described in more detail in Sect. 3.6.3. *Security* policies can be enforced with runtime models like graphs [197] or safety models [219,220] which help to manage the adaptation of the running system so the policies stay enforced in evolving system configurations. Policy checking and enforcement is further discussed in Sect. 3.3.5.

Interoperability can be achieved through adaptation by dynamic generation of mediators which translate actions of a networked system to actions of another networked system [23,130]. Reflective middleware like OpenORB [10,65] or ReMMoC [116,118] enable inspecting and adapting communication and interoperability functionalities to cope with distributed changing environments. Starlink [39,164] and OverStar [119] offer similar interoperability mechanisms by using abstract network messages and overlay networks. Dynamic code generation and middleware approaches are further discussed in Sects. 3.7.4 and 3.6.4, respectively.

Usability improvement is mainly achieved by dynamic adaptation of user interfaces. As described in Sect. 2.3, we are not going into detail about these approaches. Table 2 gives an overview of approaches relying on user interface adaptation.

To improve tool support for non-functional requirements management in general, Röttger and Zschaler [212] propose the use of context models to make the specification of non-functional measurements independent of their application in concrete system specifications. Context models allow an independent person—the measurement designer—to provide measurement definitions at different levels of abstraction. A measurement is a non-functional dimension that can be constrained to describe a non-functional property. Transformations between context models (i.e. refinement of the models) together with measurements and the context models themselves can then be used by the application designer who is able to focus on the business logic when developing an application.

Other approaches use models in combination with dynamic service selection [185], QoS mode switching [61], parameter/component variation [5,6,48], OCL constraints [137],

negotiation of QoS-aware components [175] and distributed resource measuring [283] to cope with QoS requirements. Furthermore, Calinescu et al. [45] give a detailed overview over various QoS-related approaches in general.

Change impact analysis Change impact analysis addresses detecting affected system parts in case of requirements or environmental changes as well as the consequences arising from transitions to future states (e.g. check whether predefined constraints will still hold or if the system needs to perform a structural adaptation). Furthermore, changes within models (i.e. changes of the models' elements and their relationships) are also target of impact analysis. Runtime representations of design models can be used to identify connected components in order to detect affected components, especially when using architecture models and information about service compositions. Our research identified various approaches which tackle this problem of difficult impact predictions. Sindhgatta et al. [229] introduce a framework to automatically identify possible changes in MOF-compliant models. At runtime, this framework performs fine-grained analysis to identify impacted models and their elements if model changes occur. Users have the option to apply or reject adaptation suggestions and view incremental impact of their decisions on affected artefacts. Selected changes are analyzed and propagated across the models. A similar approach of impact analysis is introduced by Waignier et al. [261] where each adaptation is first tested at the model level to check that no application property is broken. The reconfiguration is executed by a six-step control loop where events are propagated to the model level and analyzed. Adequate adaptations are then executed on an architecture description model, analyzed and applied to the running system by adding or removing components or bindings. Another approach uses graph-based application-level performance models and online history-based models instead of an autonomic control loop to analyze impacts [203] (the autonomic control loop technique is discussed in Sect. 3.4.1). Goldsby et al. [110,112] generate models of possible target systems for different environmental conditions. These models help developers to identify the functional and non-functional trade-offs between the models when adapting the system.

As a summary, relating to the problems introduced in Sect. 3.1, adaptation is mainly used to solve the problems of inaccurate predictions and changing environments. In the discussed approaches, architecture models and prediction models like queueing and Markov models are used to tackle inaccurate predictions (e.g. change impacts and performance) and changing environments. For addressing the combinatorial explosion problem, feature models are used to tame numerous possible system configurations.

3.3.2 Monitoring, simulation, and prediction

Besides adaptation, runtime models can also be used for monitoring a system, simulating future evolutions by analyzing impacts on the model level and predicting system properties like performance and reliability in case of system reconfiguration. Models enable monitoring, simulation and prediction at a higher abstraction level (closer to problem space), addressing the problem of low abstraction resulting from hand-written artefacts (see Sect. 3.1). This is especially true when using architecture models at runtime to monitor system behaviour from a global perspective. Monitoring is also related to the problem 'checking or enforcing rules or constraints' introduced in Sect. 3.1, if systems are monitored for checking specific rules or constraints. More specifically, monitoring, simulation, and prediction are closely related to adaptation since most adaptation techniques require monitoring of the running system. In addition, simulation and prediction are used as input for adaptations, for instance, in the analysis and planning phases of the autonomic control loop technique described in Sect. 3.4.1. Monitoring, simulation, and prediction are discussed in individual sections below.

Monitoring Le Duc et al. [158] propose an adaptive monitoring framework named ADAMO that tackles different quality-of-information (QoI) aware data queries over dynamic data streams, whereas QoI encompasses lifespan, precision, granularity and types of monitoring data. The framework allows access to dynamic data streams for multiple clients with different QoI needs as well as generation and configuration of appropriate elements in the monitoring system according to QoI constraints. Furthermore,the monitoring system is adaptable to resource constraints and able to manage data queries in a static or incremental way. ADAMO relies on a QoI model which formalizes data sources, monitoring queries and system resources and uses constraint solving to configure data sources according to clients needs and resource constraints.

Various approaches use the tool SM@RT (Supporting Models at RunTime) [136,231–235,278] to maintain the causal connection between the system and an architecture model. In this tool, system changes lead to corresponding architectural changes in a bidirectional way. This is achieved by creating a runtime architecture infrastructure (RAI, [198,199]) without modifying the target system, using QVT (Query-View-Transformation) bidirectional model transformations. The resulting architecture model enables monitoring and controlling of the system at a high abstraction level.

Bidirectional model transformation is also used in approaches by Giese et al. [105–108] and Vogel et al. [255,257,258] where Triple Graph Grammars (TGG) are used for supporting adaptation and architectural monitoring. In the TGG approach, a source model (low-level, for monitoring or adapt-

ing the system) is causally connected to one or more target models (high-level, specific views on the system) while synchronization is declaratively specified by TGG rules at the meta-model level for the source and target models. Lehmann et al. [160] describe meta-modelling of runtime models and present an adequate meta-modelling process.

A similar three-layered approach is proposed by Cheng et al. [62] and Garlan et al. [98]: A runtime layer observes system properties and performs low-level adaptation operations, a model layer interprets observed data with the help of analyzable architecture models and checks for constraint violations, and a task layer determines QoS requirements as high-level representations of computational needs by using desired performance profiles of required services. Architecture models are represented as graphs of interacting components, increasing the monitoring to a higher level of abstraction.

Other approaches use OCL-based monitoring with protocol state machines [127,129], consider software components as blackboxes with behavioural equivalence checking [103,177], use Petri Nets to recognize interleaved patterns of runtime events [217] or introduce dynamic monitor configuration by machine-processable quantitative and qualitative properties [31]. Furthermore, models are also used to examine execution traces in order to monitor and visualize system status [168,169].

Simulation Compared to monitoring approaches, only a few simulation and prediction approaches have been proposed in the models at runtime literature. With regard to simulation, Beltrame et al. [19] enable effective simulation and debugging through model switching at runtime, which means that less accurate system models are used for uninteresting sections of the simulation and accurate system models for system parts under more precise analysis. Tan et al. [247] use automata and code generation to monitor the system and use a simulator to run a system model together with a monitor model for design-level validation in addition to code-level runtime validation. In the approach of Redlich and Gilani [207] a performance model together with runtime performance parameters of business processes (measured by events) are fed into a simulation engine. The simulation performs better prediction than techniques using historical data because structural business process information is taken into account. With the results, new business process models can be composed which are independent from a specific business process execution environment. Weiss et al. [264] propose embedding adaptation-related information in software components. The components of the design model are enriched with self-descriptions which provide information at runtime that is necessary for adaptation decisions. An iterative development based on model transformation with simulations and model feedback enables the definition of the adaptation and the refinement of the design of the self-adaptive system.

Prediction With regard to prediction, Mos and Murphy [184] give performance predictions based on UML models created dynamically by monitoring and analyzing a running system. UML models are enhanced with performance indicators and are based on static and dynamic data to identify performance hot spots. Created models can be simulated and predictions derived for different workloads using Queueing Networks and Markov chains. Models are also used to increase the efficiency of the monitoring process by activating monitoring only for those components that are responsible for performance problems and deactivate the monitoring of the other components. Muskens and Chaudron [188] use resource models, behaviour models and scenario specifications to predict the resource consumption of executable components. Their approach is applicable for component-based embedded systems and supports diagnosis and repairing. Götz et al. [114,115] propose an approach which optimizes energy efficiency of software systems running on multiple resources. Optimization of more than one resource leads to higher energy savings since communication costs can be taken into account. Prediction of energy consumption is achieved by deriving all possible distributions of the software on a given set of hardware resources. The system is then able to reconfigure itself to achieve the lowest energy consumption possible. QoS properties like CPU consumption are the basis for optimizing energy efficiency at runtime. Epifani et al. [77] use models dealing with reliability and performance at runtime and feed a Bayesian estimator with runtime data to produce updated parameters. Analyzing the runtime models enable to predict whether a desired property will be violated by the running implementation. Nguyen et al. [192] use Coloured Petri Nets to predict performance in component-based systems.

The approach introduced by Mathis and Kerbyson [171] uses an executable SAGE (SAIC's Adaptive Grid Eulerian hydrocode) model for dynamic performance prediction of an adaptive mesh application. We will not go into detail about this approach as it focusses on High-Performance Computing (HPC), outside of the scope of this article, but the approach demonstrates the broad field of meanings and use cases of runtime models.

3.3.3 Abstraction and platform independence

An important objective of software models is raising the abstraction level, regardless of whether used at design time or runtime. With the advent of model-driven development, platform-independent models gained considerable attention. Similar to monitoring, simulation, and prediction, approaches described in this section tackle the problems of low abstraction, maintenance, and reusability resulting from hand-written artefacts and manual human interventions as described in Sect. 3.1.

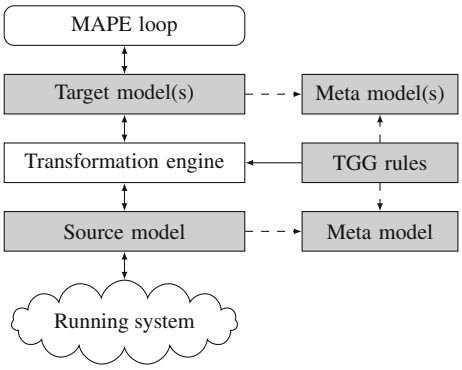

Fig. 6 Simplified view of the TGG approach introduced by Vogel et al. [257]

One attempt to raise the abstraction level is depicted in Fig. 6 which shows a simplified version of the aforementioned TGG approach [105–108,255,257,258]. It achieves higher abstraction and independence of concrete model instances by declaring TGG rules at the level of meta-models for the source and target models. This makes it easier to analyze and plan corrective actions by applying a Monitor-Analyze-Plan-Execute (MAPE) loop (see Sect. 3.4.1) to high-level target models, e.g. architecture models. Adaptive actions are propagated to the running system by a bidirectional transformation between target models and the source model which is responsible for modifying system parameters.

Other forms of raising the abstraction level are introduced by Thonhauser et al. [248,250] to support adaptation of virtual organizations and Ingstrup and Hansen [139] to support architectural reflection by creating queryable runtime models out of distributed architectures. Abstraction can also be achieved by tracing system execution with behavioural models, as described by Maoz [167].

3.3.4 Consistency and conformance checking and enforcement

Consistency and conformance checking and enforcement address the problem 'checking or enforcing rules or constraints' introduced in Sect. 3.1. In this context, consistency ensures that there are no contradictions between the different parts of a software system and/or software (development) artefacts related to the system. Conformance ensures that a software system meets a specified standard. Consistency and conformance requirements come from different kinds of sources, such as consistency with or conformance to design artefacts, valid feature combinations, other models (e.g. model synchronization) or system integrity policies, existing software standards, to name just a few sources. Models at runtime can help to check or enforce consistency or conformance requirements, as they can make model information available at runtime, either about the consistency and conformance rules or the system artefacts to be checked. They might also address the combinatorial explosion problem from Sect. 3.1: For example, using feature models at runtime, which express valid feature combinations, can help to tackle the problem of combinatorial explosion regarding possible feature selections.

Arcaini et al. [11] analyze undesirable behaviours of implementations and incorrect specifications. The conformance check is done by using Java annotations to link the concrete implementation to its formal model. The operational specification—an executable Abstract State Machine (ASM)—describes the desired system behaviour by providing a model implementation or model program of the system. An AspectJ-based monitor observes the behaviour of the application and determines its correctness with respect to the ASM specification working as an oracle of the expected behaviour. The monitor checks conformance between the observed and the expected state and produces debugging traces.

Bodenstaff et al. [37] use runtime models to ensure and maintain consistency between the running system and underlying models for inter-organizational cooperations. Consistency is ensured across different models and within models. First consistency is checked between the running system and an interpretation of the models. Consistency can then be maintained by adapting models or implementations when contradictions are detected. The check between the model and the running system is done by event logs which are said to be consistent if essential parts of the model do not contradict. The event log is thus an abstract representation of the correct runtime behaviour.

Hoehndorf et al. [131] introduce an approach using three kinds of ontologies: A task ontology serves as the conceptual model for the software, a domain ontology provides domain-specific knowledge and a top-level ontology integrates the task and domain ontologies. Types and relations of the domain ontology are combined with elements of the conceptual model through the top-level ontology. Transforming the model of an online shop into an online library then requires only a change in the domain ontology and its mappings. Instances of software models are verified against both the conceptual model and the domain ontology during runtime. This can be used to verify constraints on data that is processed by the software.

Rosenmüller et al. [211] present an approach that integrates static and dynamic feature binding seamlessly, ensuring consistency so that only valid feature combinations can be selected at compile and runtime. At deployment time, features are declared to be bound statically or dynamically. Features that are always bound together are merged into a binding unit. A feature model is then transformed according to the binding units, yielding a feature model that encom-

passes only dynamic variability. This model is used to verify a given dynamic feature composition before it is used.

Other approaches check conformance and consistency in user interface adaptation [14,79], by recording messages exchanged between components and services [40,156] and by clustering normal data to detect data anomalies [92].

3.3.5 Policy checking and enforcement

Policy checking and enforcement encompass models used at runtime to cope with policies, such as (real-)time constraints, access control and security regulations, or other compliance rules, thus tackling the problem 'checking or enforcing rules or constraints' introduced in Sect. 3.1. Different runtime models help to satisfy such policies: Safety models describe safety properties of systems, role-based access control (RBAC) models define access management policies, and Real-Time-UML (RT-UML) models support the modelling of timing constraints to be preserved at runtime.

Holmes et al. [132–134] introduce an approach where monitored information is interactively interpreted and analyzed with respect to compliance to regulations, using models as first-class citizens. These models describe a service-based system and its system requirements and are used at runtime to trace back violations. This way this approach also addresses the problem 'error localization' from Sect. 3.1. Information described in models can be queried and developers are able to perform modifications on-the-fly and add new model versions to the running system so newly created model instances can immediately use the corrected version. Multiple versions of models are maintained in parallel so that old model versions can be used until all of their model instances are deleted or migrated to new versions. Modifications are done via web services while consistency is maintained by artefact relationships and compliance annotations expressed via dedicated Domain-Specific Languages (DSLs). Services can be generated with basic create, retrieve, update, delete, and query operations.

In the approach of Schneider and Trapp [219,220] dynamic changes in a system or environment are reflected based on safety runtime models. They consist of configuration models (describe configurations, required/provided services of a component), an adaptation model (rules to determine the best configuration), models describing non-functional service properties (required/provided QoS and mappings), and safety models (required/provided safety properties and their mappings). Required and provided safety properties are matched throughout a composition tree holding all required services.

Hummer et al. [138] realize identity and access management (IAM) in the context of service-oriented architectures by using model information at runtime. RBAC models are specified by using a DSL and are mapped to source code artefacts of the software platform via automated model transformations. Business processes—annotated with DSL elements—are instrumented with special activities to ensure compliance to IAM policies at runtime.

RBAC policies can also be modelled as graph to reason on architectural changes and maintain enforced policies [197]. A mapping between RBAC policies and a component-based architecture allows to transform policies into an architecture enforcing them and to analyze the policy enforcement on the architecture. Furthermore, an access control reasoner can listen for architectural changes in order to manage the adaptation of the running system so the policies stay enforced in the system.

Halfond and Orso [125] use a model-based approach to detect illegal queries before they are executed on the database. A model of legitimate queries that could be generated by the application is automatically built at design time. At runtime, monitoring is carried out to inspect dynamically generated queries and check them against the built model. In case of violation, queries are prevented from executing on the database and reported to developers and administrators. Figure 7 shows an SQL query model extracted from [125] which defines allowed login queries for guests and users.

In the approaches of Rammig et al. [206] and Zhao et al. [279–281] a model-based acceptance test is introduced where verification is performed at the level of RT-UML models representing the systems under consideration. Checked properties are defined by the notation of RT-OCL (Real-Time Object Constraint Language) while the overall system behaviour is specified by RT-UML statecharts. RT-UML models and associated RT-OCL constraints are transformed into Kripke structures and Büchi automata and then stored in a repository. At runtime, if a component replacement is requested within the component architecture, the operating system sends a

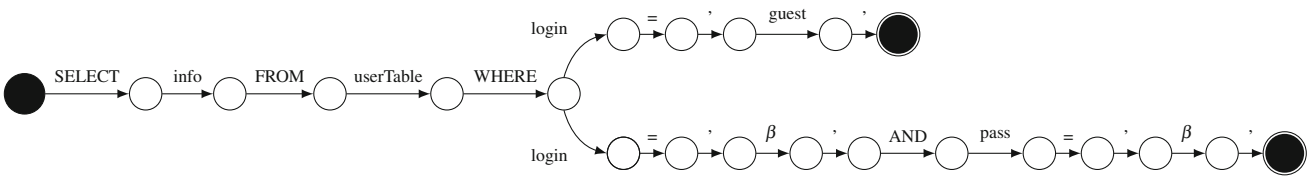

Fig. 7 Example SQL query model by Halfond and Orso [125] to prevent illegal database queries

request to a verification service which starts verifying by fetching related Kripke structures and Büchi automata from the repository. The service answer is *Yes*, *No* or *Unknown* as decision about accepting or rejecting the requirement for substitution.

Other approaches perform model-based real-time and schedulability analysis for embedded systems [123], manage physical network equipment [176], ensure security in an agent-oriented way [272], realize continuous quality assurance of Web services [15], use a model-based timing analysis engine deciding whether a configuration is feasible to be executed [242], use aspect-oriented programming to integrate security controls within the target system [7], and verify design models against high-level security requirements such as secrecy and authentication [18]. Threats can also be modelled as sequence diagrams which are compared with system traces to detect undesirable threat behaviour [263].

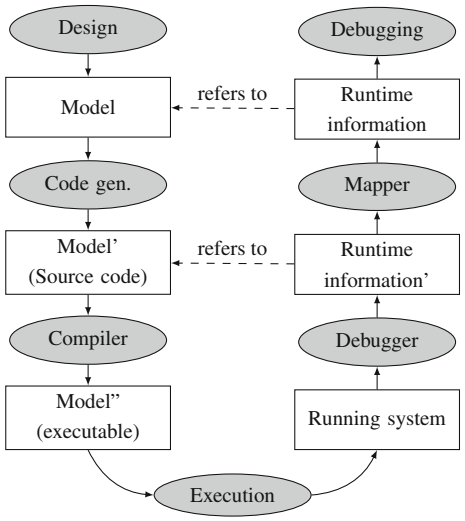

Fig. 8 Model layer mappings introduced by Graf and Müller-Glaser [122]

3.3.6 Error handling

Runtime models can be used for more efficient error handling in form of debugging, fault localization, tracing, self-healing and test case limitation and generation. Hence, problems like 'error localization', 'inaccurate predictions', and 'changing/heterogeneous environments' from Sect. 3.1 are addressed by approaches reported in this section.

State machine models and workflow models are usable for many of these scenarios because of their expressive nature in declaring execution and data flows which can easily be checked against given execution traces, obtained by emitted events or execution logs. Regarding testing, approaches exist which use state machines for automatic iterative test case generation [273] and claim refinement models [265] to estimate the probability of encountering predefined system states, thus reducing needed test cases.

Simmonds et al. [228] propose an approach for enhanced recovery in web service applications. Behavioural correctness of conversations within the web service system is monitored and checked against correctness properties specified by developers. Such properties are transformed into finite state automata which enable conformance checking of execution traces of web services described in BPEL. While traversing the automata, going back and re-planning of the conversation in case of a fault is possible: Occurred actions are revoked until an alternative behaviour of the application is possible (i.e. backtracking the states of the automata).

Fuentes and Sanchez [90] implemented a dynamic model weaver that can be used for running aspect-oriented models where aspects are woven and unwoven during model execution, leading to reconfiguration of the system and taming combinatorial explosion by well-defined adaptation points. Designers can also interact with the dynamic model weaver, loading and unloading pointcuts that activate and/or deactivate aspects which results in a more interactive model simulation. Such weavers enable the possibility to test and debug models before moving into an implementation which tackles the problem of inaccurate predictions and error localization described in Sect. 3.1.

Graf and Müller-Glaser [121,122] (see also Kordon [153]) describe an architecture that allows for the definition of various debugging-perspectives independent of the actual execution-platform. Figure 8 shows their idea of mappings between different model layers. The basic idea is to extract runtime information out of the executed binary from the target platform and transport this information back through the abstraction layers to model level. Obtained runtime information on model level can then be used to visualize the internal state of the executable. To achieve this, they extend the UML meta-model with meta-classes that allow storage of data acquired by the mapper. This approach tackles the problem of error localization as well as raising the abstraction level as introduced in Sect. 3.1.

Zeng et al. [275] follow a similar approach by targeting a model as the object of testing. A Graphical Debugger Model (GDM) is established which performs the role of a server interacting with the executable code. Once the application is running, GDM starts animating the system's running behaviour at model level with help of an event-driven finite state machine which waits for commands sent by target embedded code. If received actions are not consistent with system requirements, a bug is reported. A similar approach is tackled by Spieker [239], Hooman and Hendriks [135] where a model of the desired system behaviour is modelled by a state machine. At runtime, the system state is compared to the model, possibly leading to errors. As a consequence, diag-

nosis and recovery phases are applied and corrective actions are executed, thus solving the problem of finding errors in such dynamic systems.

Nordstrom et al. [195] introduce fault localization in workflows with future state determination, enabling impact analysis when dealing with unforeseen error occurrence. They use a workflow execution engine simulator running in parallel with the system. A workflow model represents the current state of the workflow by using a synchronizer or observer monitoring the progress of the workflow execution. In case of an error, the model is updated and a simulation algorithm determines future states of the workflow assuming no external intervention or future faults. Furthermore, jobs within the workflow model are annotated as desired or undesired to be used as metric to determine the relative desirability of a partial workflow.

A middleware-based approach is introduced by Haberl et al. [124] where a system model is subsequently transformed into executable distributed code. The middleware is used for interaction within the system by acting as global communication router between the generated software components. As a result, the middleware is able to access the internal state of the system and store internal data of components for later debugging. Captured runtime data enable debugging at a higher abstraction level by mapping them back to the model while the handling of error occurrences is simplified by shifting the analysis to the central middleware entity.

Other model-based approaches use model analysis to reduce or generate test cases [265,273], monitor events with concurrent Petri Net models [52], use stop-and-restart strategies of malfunctioning services [243], gauge effects of partial failure in distributed systems [271] and classify errors based on quality attributes [187]. Porcarelli et al. [205] realize fault tolerance by using Petri Net models to drive decisions about reconfiguration actions in distributed systems. Krasnogolowy et al. [154] use story diagrams to debug applications with the help of breakpoints, step-wise execution, control flow visualization, variable inspection, variable modification and remote debugging. Another approach is introduced by Occello et al. [196] to detect errors within adaptation processes. They propose a runtime model which enables the verification of dynamic adaptation as described in Sect. 3.3.1. An overview of using architecture models for problem diagnosis and repair is given by Garlan and Schmerl [96,97].

3.3.7 Overview of approaches

Table 2 gives an overview of approaches according to the classes of objectives introduced in Sect. 3.3: Adaptation, monitoring/simulation/prediction, abstraction and platform independence, consistency/conformance checking and enforcement, policy checking and enforcement and error handling.

3.4 Techniques

We classified our identified techniques used in combination with runtime models into *autonomic control loop* (analyze running system and plan corrective actions), *introspection* (extract data from a running system), *model conformance* (ensure conformance and consistency against models), *model comparison* (check differences between two models), *model transformation* (change representation of models), and *model execution* (execute models with operational semantics).

3.4.1 Autonomic control loop

In the existing literature, we identified three strategies to address the five adaptation scenarios described in Sect. 3.3.1: Autonomic control loop, structural adaptation and parameter adaptation. Structural and parameter adaptation are often used in combination with the autonomic control loop strategy, so we decided to include them in this section as well. The characteristics and relationships between these strategies are discussed in the following paragraphs.

Autonomic control loops, as in the approaches of Waignier et al. [261] and Pienaar et al. [203], are a key concept when adapting systems at runtime [198]. The idea originates from the automatic computing research community to enable self-management of systems according to desired goals [147]. Figure 9 illustrates the idea of autonomic control loops which is to measure system parameters, analyze them, plan corrective actions if necessary and execute them. This procedure is known as Monitor-Analyze-Plan-Execute loop (MAPE/MAPE-K, with a shared knowledge component [147]) or Collect-Analyze-Decide-Act loop [73]. The planning component of the loop needs to identify alternative configurations (that is, an arrangement of system parts) that satisfy current contextual requirements and constraints. There exist formal methods to compute solutions to such configuration problems and to check whether given requirements are not violated [41].

Dubus and Merle [74] use models within this control loop as abstract representation of the system to reconfigure appli-

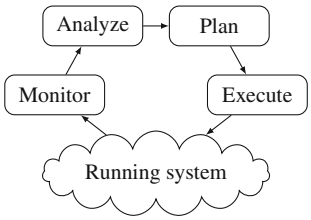

Fig. 9 Idea behind an autonomic control loop

cations. This model evolves with the system and describes deployment entities and application components, helping to plan adequate reconfiguration actions. A similar approach is introduced by Garlan et al. [95] where their framework uses abstract architecture models to monitor a system, check constraint violations and trigger global- and module-level adaptation if necessary. The abstract architecture model offers a global perspective on the system and exposes important system-level properties and integrity constraints. Gamez et al. [91] use additional feature models in the control loop to represent potential middleware configurations at runtime, shifting reconfiguration to the middleware level. The set of actions that must be executed to pass from one valid configuration to another is specified in an OWL-S plan model compliant with feature and architecture models. Other approaches include machine learning [253] and execution trace analysis [189] in the automatic control loop. One benefit of such autonomic control loop lies in the reduced need for manual human interventions which often lead to low abstraction, maintenance, and reusability, as pointed out in Sect. 3.1. Another advantage are the improved self-management capabilities for running systems (self-configuration, self-optimization, self-healing, self-protection [147]).

Within the autonomic control loop, the process of adaptation itself is accomplished either by *structural adaptation* (system component variability) or *parameter adaptation* (modification of variables to alter program behaviour). Floch et al. [83] apply structural adaptation by plugging in different component implementations whose externally observable behaviour conforms to a specific component type. Together with parameter adaptation, a mixture of coarse-grained (structural) and fine-grained (parameter) adaptation is achieved. Approaches by Amoui et al. [8,9] and Derakhshanmanesh et al. [71,72] use the framework GRAF (Graph-based Runtime Adaptation Framework) which also supports both adaptation strategies. Structural adaptation is realized by replacing a block of the control flow with the execution of an alternative behaviour with equivalent pre- and post-conditions at runtime. Blocks are replaced in a behaviour description that is part of the runtime model and are executed by a model interpreter when the control flow decides about the interpretation method. For parameter adaptation, the application flow is redirected to an alternate action at a Turning Point which is a branch in the control flow with a well-defined decision criterion. Such a criterion is a special variable that can be changed in the runtime model, propagating values back to the adaptable software. Another approach using both parameter and structural adaptation is introduced by Caporuscio et al. [48] where runtime performance management is achieved by adapting the number of threads (parameter) or adding/removing component or connector instances (structural).

3.4.2 Introspection

Introspection deals with extraction of runtime system data in order to apply model-based and control loop techniques on the analyzed behaviour. We subdivided introspection mechanisms into three different groups:

- Event log checking
- Instrumentation
- Management API

Event log checking accomplish the data extraction by scanning event log entries emitted by the system under observation. Such event log entries have to follow a defined structure do be processed externally and in further consequence to be related to runtime representations of models. The previously mentioned approach of Bodenstaff et al. [37] uses this technique to check a model against the running system. The event log is assumed to be consistent with the running system and also with a model if essential parts of the model do not contradict. In other words, it is assumed that the event log reflects a correct representation of the running system. The challenge in this approach lies in identifying relevant parts of the event log and abstracting them to enable consistency checking between the running system and a model. To achieve this, either the system is adapted so that emitted events have the proper format or the necessary format is reconstructed from raw event logs.

In case of *instrumentation*, monitoring functionality is inserted into the system under observation. Insertion of monitoring code requires the source or bytecode to be available and is often realized by using aspect-oriented programming to weave data extraction code into the application. Extracted data can then be related to runtime representations of models to enable reasoning on a higher level of abstraction. On bytecode level, Hamann et al. [128] propose monitoring of JVM hosted applications by using a platform-aligned model (PAM) as link between the platform-specific model (PSM) and the platform-independent model (PIM). A PAM is iteratively designed through assumptions which are stated, checked, and refined. A first iteration of the PAM can be described in a declarative way, reverse engineered out of the implementation or derived from component specifications.

Nierstrasz et al. [193] use instrumentation at runtime in combination with an Abstract Syntax Tree (AST) model as abstraction above the bytecode level. They introduce the concept of links to be set as annotations to AST nodes which are transformed by a compiler plugin before execution. This results in code to be inserted in the program at nodes where links are installed. Such links can also be installed by other links or manipulated by themselves. A similar approach is tackled by Halfond and Orso [125] where code instrumentation adds calls to a monitor to check for illegal SQL queries

at runtime. An example validation model for SQL queries is depicted in Fig. 7. Other approaches realize instrumentation by aforementioned aspect-orientation [261] or by planting special activities into WS-BPEL processes to enforce compliance to IAM policies at runtime [138]. All approaches tackle the problem of low abstraction introduced in Sect. 3.1 by shifting analysis to the model level.

In case of *Management APIs*, an interface on the target system exists which allows extraction of runtime state information. An example of such an interface are reflection APIs of modern programming languages which allow to query and modify the structure of a computer program at runtime. A reflective architecture consists of two causally connected layers: A base-level realizes the functional aspect of the system, whereas a meta-level realizes the non-functional aspect of the system and carries out the evolution on a reification of the design information [54,163].

Reflection is used in reflective middleware like OpenORB [10,65] or ReMMoC [116,118] to inspect and adapt communication and interoperability functionalities to cope with distributed changing environments. The base level of OpenORB consists of interfaces, components, and bindings, whereas the meta-level is a type repository containing meta-information such as interface definitions, component definitions, and binding definitions [64]. Such reflective middleware is also able to utilize aspect-oriented techniques to adapt or re-order advice behaviour to further increase adaptation capabilities of the middleware architecture [120,252] (see Sects. 3.6.3 and 3.6.4 for details to these architectures).

The key issue is to maintain a causal connection based on management APIs provided by target systems. Such an approach is followed by Huang et al. [136] and Song et al. [231–235] by implementing a synchronization between an architecture model and the system state with respect to consistency, non-interference introspection, effective reconfiguration and stability. The authors utilize code generation for wrapping the low-level APIs, use a system-model adapter to support reading and writing system state in a model-based way and thus solve the problem of low abstraction by manipulating the system at the model level.

3.4.3 Model conformance

Model conformance addresses the conformance and consistency of data and processes according to models. Model conformance is closely related to the previously described objectives of consistency and conformance, policy checking/enforcement and error handling (see Sects. 3.3.4, 3.3.5 and 3.3.6, respectively) since extracted runtime information is checked against models to trace constraint violations and system behaviour. Model conformance requires introspection mechanisms introduced in Sect. 3.4.2 in combination with runtime representations of the models, e.g. by using the common XMI (XML Metadata Interchange) format. Tombelle and Vanwormhoudt [251] give an overview of introspection and model scripting techniques to process these runtime model representations. Examples of ensuring conformance and consistency against models are graph-based approaches (Petri Nets, state machine model), the usage of Linear Temporal Logic (LTL) and the enforcement of OCL constraints.

A combination of the graph-based strategy and LTL is used by Rammig and Zhao et al. [206,279–281]. LTL formulas are derived from RT-OCL constraints and transformed into Büchi automata. By using such an automaton in combination with the source code and an abstract system model (expressed as state machine), model checking shows whether an execution trace conforms to the system model (consistency checking) and whether a partial state space of the system model conforms to the Büchi automaton (safety checking). A similar aforementioned approach of combining LTL and Abstract State Machines (ASMs) is introduced by Arcaini et al. [11]. Both approaches address the problems of low abstraction and error localization during runtime by using models at runtime.

Another approach using LTL is followed by Zhang and Cheng [277]. They describe the need of building a model for a source domain (the source model) and a model for a target domain (the target model). The source and target models should be verified against local requirements for the source and target domains, respectively. This verification is expressed via LTL formula to reason about adaptation mechanisms in wireless networks. Other approaches include ReMMoC [116,118], Starlink [39,164] and OverStar [119] which check conformance of communication processes to ensure interoperability (see Sect. 3.6.4 for details to these approaches).

3.4.4 Model comparison

Model comparison addresses the comparison of two models, especially to yield information about their differences and needed delta-operations to transform one model into another model. An often referenced tool for providing model comparison functionality is EMF Compare (Eclipse Modelling Foundation), although alternatives like KMF (Kevoree Modelling Framework) are developed to improve performance, reduce memory footprint, and eliminate library dependencies [69,86]. Model comparison is mainly used to calculate operations needed for transitions between system states.

An adaptation approach using model comparison is introduced by Fleurey and Morin et al. [82,180,181]. The key idea is to produce a reference model of the running system using aforementioned introspection mechanisms. The resulting model represents the current configuration which is then compared to a model containing the new configuration. The comparison yields another model that specifies the differ-

ences and the similarities between the models. By analyzing this model, reconfiguration commands are created which add and/or remove bindings and/or components to reflect the requested adaptation. The same comparison principle is followed by Waignier et al. [261] where a new architecture description model is compared to the model of the running system to identify reconfiguration actions and calls to the platform-specific API. Both approaches address the problem of requirement evolution and changing environments introduced in Sect. 3.1 by adapting the system according to the new requirements.

3.4.5 Model transformation

Model transformation is used to change the representation of a source model into another form, called the target model. Such transformations aim to reduce errors through automation while ensuring consistency between the source and the target model. They are generally divided into Model-To-Text transformation (converting a model into a textual representation, e.g. source code) and Model-To-Model transformation (output is another model). Established methods to define such model transformations are QVT (Query-View-Transformation) and TGG (Triple Graph Grammar). We already pointed out approaches using TGG [105–108,255,257,258] (see Fig. 6) and ones that use QVT [136,231–235]. The approaches tackle the problem of low abstraction by transforming low-level representations of the system into one or more models of higher abstraction to simplify monitoring and controlling of the system under observation, mainly by targeting architecture models as transformation output to obtain a basis for system-wide reasoning.

3.4.6 Model execution

By the time of writing, it is not entirely clear how model execution fits into the models at runtime paradigm since its relationship to dynamic reflection and introspection needs (see Sect. 3.6) is debatable. It is a recurring question at the Models@run.time workshop [27]. We decided to include model execution as separate technique since changes to an interpreted runtime model directly imply changes in the behaviour of the running system - this property clearly falls into the definition of a *causal connection* (see Sect. 1). Furthermore, we feel obliged to follow a holistic approach when conducting a literature review, so we decided to include this topic in the article, although we are aware that the ongoing discussions are not for no reason. Also, this technique is strongly related to the workflow community since model execution is a common procedure in workflow execution engines.

Model execution attends to direct execution of models containing operational semantics. In other words, one representation serves as both the model and the program which are necessarily at the same abstraction level [109]. Muller et al. [186] introduce weaving of executability with static type checking and support for genericity into meta-data languages. The resulting combination eases simulation and testing of operational semantics of meta-models. The OMG standard *fUML* introduces executable semantics to a subset of UML where models can also be used at runtime in combination with execution traces [172,173]. A popular example of executable models are state machines which are used throughout our identified literature. They allow to specify execution flows at the model level independent of the concrete implementation, solving the problem of low abstraction resulting from hand-written artefacts.

Arcaini et al. [11] use ASMs as executable specification and oracle of the expected system behaviour. A monitor checks conformance between the observed system state and the expected state by executing the state machine. The aforementioned GRAF framework [8,9,71,72] achieves adaptation by redirecting the control flow of the software to a model interpreter component at points where the need for adaptivity is expected. When such an interpretation point is reached during execution, the model interpreter executes associated behaviour described in a runtime model.

In the approach of Pleumann and Haustein [204], domain models are directly executed to realize dynamic web applications. Class diagrams with OCL annotations are the central elements to drive database access, user interface generation, and business logic interpretation in a three-tier architecture. Database access is realized by a persistence layer to XML files or existing databases. Business logic is interpreted with the help of UML state machine models linked to elements of the UML class diagrams. A model-driven runtime combines the class diagrams, state machine models, and special HTML templates to a working three-tier web application.

Other approaches realize executable task models describing a task life cycle [38], use model execution to guide test data generation [273], execute integrated behaviour models [149], interpret models to ease development of multi-tier applications [226], interpret state machines of model-based components [249], execute models to support adaptation and service delivery validation [50,183] and support user interface development using executable models containing static structure, dynamic state and execution logic information [159].

3.4.7 Overview of approaches

Table 3 gives an overview of approaches related to the different techniques introduced in Sect. 3.4: Autonomic control loop, introspection, model conformance, model comparison, model transformation, and model execution.

Table 3 Overview of approaches related to the identified techniques

Technique	Related approaches
Autonomic control loop	[8,9,48,71,72,74,83,91,95,101,189,203,253,261]
Introspection	*Event log checking*: [37]
	Instrumentation: [125,128,138,193,261]
	Management APIs: [54,64,136,163,231–235]
Model conformance	[11,39,116,118,119,164,206,251,277,279–281]
Model comparison	[82,180,181,261]
Model transformation	[105–108,136,231–235,255,257,258]
Model execution	[8,9,11,38,50,71,72,149,159,172,173,186,204,226,238,249,273]

Table 4 Overview of kinds of models and their purpose

Kind of model	Objectives of runtime models	Usage
Abstract syntax tree	Adaptation	[193]
Architecture model	Adaptation, monitoring	[83,101,126,136,231–235]
Aspect model	Adaptation, error handling	[82,90,180,181]
Business process model	Adaptation, policy checking and enforcement	[88]
Context model	Adaptation, monitoring	[140,143,223–225]
Feature model	Adaptation, consistency	[3,55–58,91,140,143,148,200,211]
Goal/requirements model	Adaptation	[4,21,22,111,141,142,214,227,260,266,274]
Observation model	Abstraction, monitoring	[276]
Performance model	Adaptation	[47,162]
RBAC model	Policy checking and enforcement	[138,197,244]
Safety model	Policy checking and enforcement	[219,220]
State machine model	Adaptation, conformance, error handling	[11,17,135,275]
Task model	Adaptation, monitoring	[68,223–225]

3.5 Kinds of models

In this section, we focus only on the kinds of models used in the publications after applying our selection criteria of the literature search. Many more kinds of runtime models exist that we do not discuss in detail, like Petri Net models [52,146,192,217,221,268,269], policy models [15], Probability Fuzzy Cognitive Maps [262], quality models [187], physical models [2,70], contracts [33,60,190], Markov chains [81,104], Queueing Networks [51,104,184] and domain-specific languages in general [1,46,214,267]. The diverse kinds of models demonstrate how elastic the term *model* truly is.

We want to point out one kind of model which is frequently used in recent research: *Goal models*. Goal models describe the relationships between a system and the environmental context [274]. With contextual information, goal models enable to assess candidate solutions against high-level criteria for stakeholders to analyze system-wide trade-offs [260]. Using goal models (or requirement models) at runtime takes the idea of runtime models one step further towards problem space. Welsh et al. [266] use goal models to realize requirements-aware systems by checking assumptions made at design time. If such assumptions do not hold, system-wide adaptations are triggered to enable alternative goal realization strategies. Goldsby et al. [111] use multiple goal models containing requirements specified by different developers at different levels of requirements engineering. Alférez and Pelechano [4] use requirement models in combination with various other models to support dynamic evolution of context-aware systems: An architecture model which describes the architecture of the system, a context model holding context knowledge, tactic models which describe strategies for preserving the requirements at runtime and a variability model describing dynamic configurations of the system. Silva Souza et al. [227] propose Awareness Requirements (*AwReqs*), requirements that inform about the success or failure of other requirements. Such AwReqs are represented in a formal language and can be directly monitored by a requirements monitoring framework. Furthermore, they propose a graphical representation that allows to include the AwReqs in goal models to improve the communication among system analysts and designers. Other approaches using goal/requirement models are listed in Table 4.

We decided to list our selected kinds of models in Table 4 instead of going into detail about every other kind of model, as the majority of them has already been introduced above

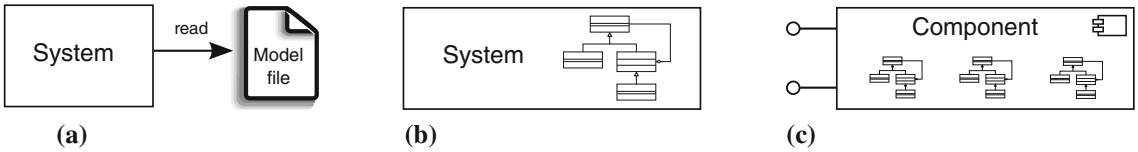

Fig. 10 Model access examples when using a monolithic architecture. **a** Access a model file, **b** access an in-memory model, **c** component with embedded model

when discussing the respective approaches. The table provides an overview about the purpose of each kind and their usage in our identified literature. We classified the purposes according to our identified objectives of runtime models. Kinds of models not explicitly mentioned in previous sections are business process models, context models and observation models. *Business process models* describe sequences of business activities and are used to check workflow conformance and guide adaptation through extensions to BPMN (Business Process Model and Notation) elements [88]. Ontology-based *context models* are used to model context information to enable monitoring of context changes and simplify adaptation decisions. Serral et al. [223–225] use such context models in combination with task models to describe user behaviour patterns and automate user routines. An *observation model* is an abstraction of the system which consists of a hierarchy of monitored entities and includes a collection of metrics used to measure the performance of business operations [276].

3.6 Architectures

Processing models at runtime requires an architecture with introspective capabilities to extract relevant runtime information and to relate the collected data with models. This is very similar to reflection where the system under observation is able to access its own structure which is causally connected to it (modifications to this self-representation are reflected in its own status and computation) [166]. Models at runtime differ from reflection mainly in terms of abstraction: While reflection is more *solution space-oriented*, models at runtime operate on a higher level of abstraction towards *problem space* [34]. At the model level, model-driven techniques can then be utilized at runtime which is not considered in traditional model-driven approaches [20,34,218].

While investigating our literature search results, we identified five main types of architectures when processing models at runtime: Monolithic, local dataflow, model-aware middleware, communication middleware, and repository architectures. They differ mainly in terms of model access and level of dynamics like adaptation or simulation capabilities. Monolithic architectures encompass functionality in a single system component without separation of concerns. Local dataflow architectures use inter-process communication or in-process communication between different components or threads to gain flexibility through modularity. Middleware architectures enable distributed communication and use middleware components for additional functionality such as improving control, monitoring, and logging. Repository architectures enable concurrent access to model data stored in a central repository and archiving of different model versions. We discuss each of the architectures in detail in subsequent sections.

3.6.1 Monolithic architecture

In a monolithic architecture, the whole system functionality is captured in a single unit without separation into multiple system components. In case of runtime models, we speak of a monolithic architecture if a runtime model is either locally accessed and/or manipulated by the running system itself or is directly integrated in a software component, e.g. as woven code or as meta-data of an assembly file. The missing loose coupling of system components leads to a simple system design, but lacks adaptability, scalability and reusability. Therefore, such an architecture is often used in combination with direct model execution or interpretation (see Sect. 3.4.6) when dynamic aspects like adaptation, simulation and debugging can be neglected. Figure 10 shows three examples of runtime model access in monolithic architectures: (a) a system which consumes a local model file (e.g. in XMI format), (b) a system which has access to an in-memory model (e.g. through woven model code) and (c) a component which has embedded model information with a provided communication interface. Example approaches using monolithic architectures are introduced by Thonhauser et al. [249] (interpret state machines of model-based components) and Elkorobarrutia et al. [76] (use state machine models within components as basis for self-healing mechanisms).

3.6.2 Local dataflow architecture

Local dataflow architectures are used to decompose a local task into several subtasks with lower complexity, thus compensating disadvantages of monolithic architectures introduced by missing loose coupling. Dataflow between local subtasks can be implemented by inter-process and in-process communication mechanisms like sockets and threads, respec-

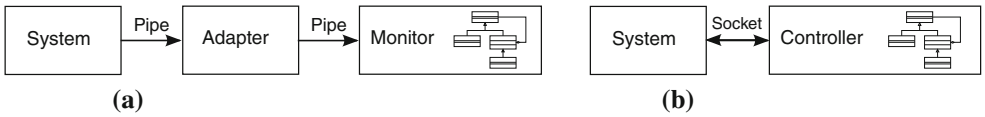

Fig. 11 Inter-process communication to separate the model from the system under observation. **a** Pipes and filters communication, **b** socket communication

tively. The Pipes and Filters architectural style [43] is also a possibility to realize such an architecture [199], although we could not find a concrete approach in our literature search which uses Pipes and Filters. Figure 11 shows two examples of pipes and filters and socket communications. In Fig. 11a, the pipes and filters architecture is used for subsequent transformation of observed system information into data to be visualized in a model, enabling monitoring and simulation through a reusable component. Figure 11b shows the communication between a system and a model-aware controller part through sockets. Since socket communications are bidirectional, this architecture qualifies for dynamic approaches with adaptation capabilities, as pointed out by the double-headed arrow in Fig. 11b. A socket-related approach is introduced by Hooman and Hendriks [135] where a model executor is integrated into the controller part to check the runtime system state against a state machine model, leading to corrective actions if errors are detected. Götz et al. [114,115] also use this kind of architecture to realize energy auto-tuning by optimizing multiple resources in distributed systems.

3.6.3 Model-aware middleware architecture

Middleware architectures enable distributed communication and provide additional functionality such as improving control, monitoring and logging. A model-aware middleware architecture consists of a central entity responsible for processing model information and providing model access services within a distributed system. Such architecture helps to maintain monitoring and adaptation mechanisms at a central place, increasing reusability while compensating limitations of local machine boundaries existing in monolithic and local dataflow architectures. Figure 12 shows an example model-aware middleware which monitors and controls a system while collecting information from other data sources (e.g. sensors or manual input data to drive reconfiguration actions).

Since there exists a variety of middleware approaches, we will not go into detail about all of them. Instead we pick out approaches covering the class of model-aware middleware architectures very clearly and will mention other approaches in short form.

One previously mentioned approach is introduced by Taconet et al. [246] which makes dynamic use of new context data collectors. Their middleware CA3M (Context-

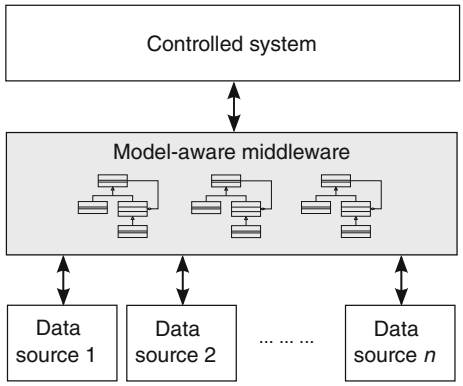

Fig. 12 Model-aware middleware for monitoring and controlling a system according to input data

Aware Middleware with context-awareness Meta-Model) uses context-awareness models present at runtime which are updated to fulfil new application requirements. CA3M dynamically constructs bridges between applications and distributed collectors to realize monitoring elements of the context-awareness model. Instances of the models are updated at runtime, demonstrated by the authors as new plugins are added to a mobile-chat application. CA3M offers two kinds of interactions: Observation mode (application initiates exchange of collector data) and notification mode (the other way around, periodically or threshold-based). By using the CA3M middleware, the authors tackle the problem of changing and heterogeneous environment pointed out in Sect. 3.1.

The MADAM middleware owns a planning framework that subsumes and automates adaptation decision-making in reflective component-based adaptive systems [5,6]. Variability of the adaptive system is modelled as variation points where component compositions are selected to build an application. Utility functions evaluate the user benefit of a component at a variation point in the composition. Each component has a type that defines its signature and externally observable behaviour. Components can be plugged in at a variation point if the type conforms with the type specified for the variation point. They are annotated with non-functional metadata (according to a given QoS model) which is associated with component ports to specify the services provided or required at ports. This metadata is then used to maximize the utility of the application: For example, minimum resource consumption, maximum resource consumption (likely to be best performance), or maximum efficiency (ratio utility to resource consumption).

Another approach is introduced by Haberl et al. [124] where a system model specified in COLA (COmponent LAnguage) is subsequently transformed into executable distributed code. A middleware acts as global communication router between generated software components and is thus able to access internal system state, store internal data of components, and record input/output values. These data are fed back to the model, allowing debugging on a higher abstraction level and thus solving the problem of reduced level of abstraction resulting from hand-written artefacts as described in Sect. 3.1.

Other approaches use feature models to represent potential middleware configurations at runtime [91] or use architecture models to control adaptation and a middleware to identify all alternative components pluggable into component frameworks [83].

3.6.4 Communication middleware

A communication middleware focusses on abstracting away complexity of network communication between distributed components. As seen throughout this article, the usage of services is well established in combination with runtime models. Many middleware-based approaches use services as primary communication method between distributed components. The approach introduced by Rammig and Zhao et al. [206,279–281] use service calls of a model-based service to verify reconfiguration plans. The approach is depicted in Fig. 13 in simplified form. Assuming a substitution of component D with E of a given application, the question arises if real-time constraints will still be satisfied and if the system still keeps safety and consistency after the replacement. To answer this question, RT-UML models and associated RT-OCL constraints are transformed into Kripke structures and Büchi automata which are accessible via a service. At runtime, if a component replacement is requested within the component architecture, the Real-Time Operating System (RTOS) sends a request to this verification service containing system state and timing constraints. The verification service then fetches related Kripke structures and Büchi automata and immediately starts on-the-fly verification. The service answers *Yes*, *No* or *Unknown* as decision about accepting or rejecting the component substitution.

A key challenge of distributed communication is interoperability between different network participants. Grace et al. [116,118] propose a reflective middleware ReMMoC that allows mobile clients to be developed independently of both discovery and interaction mechanisms of web services. ReMMoC uses a binding framework for interoperation with different interaction types (e.g. SOAP RPC, event subscriber) and a service discovery framework for discovering services using a range of service discovery protocols. Similar interoperability goals are pursued by the frameworks Starlink [39,164] and OverStar [119]. Starlink uses abstract network messages paired with coloured automata that represent the required interoperability behaviour between protocols. OverStar utilizes self-managing overlay networks with open access to the overlay nodes, which enables additional flow logic. Another approach is proposed by Bencomo et al. [23] where mediators are dynamically generated to translate actions of one networked system to actions of another networked system. Such code generation strategies are discussed in Sect. 3.7.4.

More approaches exist which use aforementioned model-based adaptation for dynamic service selection [49,88], use a diagnose service to request monitoring information [240, 241] and enforce RBAC constraints in service-based business processes [138].

3.6.5 Repository architecture

Repository-based approaches use a central storage of models or model-related data. The advantage of the repository-based approaches lies in the controlled access of model instances as well as the centralized storage of model evolution information. Repositories are used to establish a distributed architecture and ease access and management of model data by providing standardized interfaces for model manipulation. Figure 14 shows a repository architecture which enables concurrent model access and versioning. Repository updates can either be applied by manual actions or other systems which have writeable access to the repository.

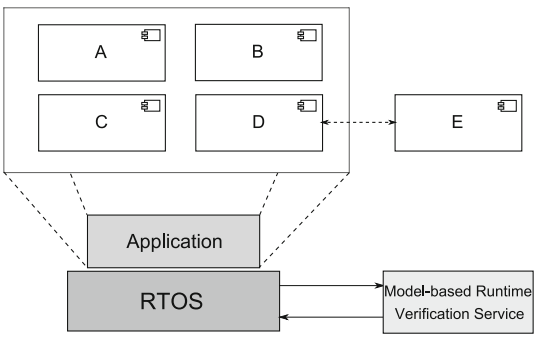

Fig. 13 Simplified view of the case study introduced by Zhao et al. [280]

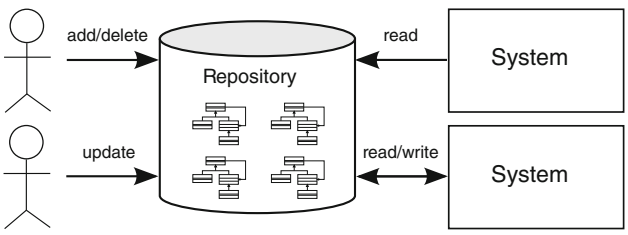

Fig. 14 Repository architecture which enables concurrent model access and versioning

The aforementioned approach by Holmes et al. [132–134] interactively interprets and analyzes monitored information with respect to compliance to regulations, using models as first-class citizens. Models are stored in a central MORSE repository (Model-Aware Repository and Service Environment) and can be queried and manipulated via web services while the system runs. Furthermore, the repository is able to store multiple versions of models so that old versions can be used until their instances are deleted or migrated to new ones. Thus, the advantage of the repository lies in hot-plugging new model versions so newly created instances are instantly able to use them. However, the challenge of such a repository approach lies in maintaining consistency. The authors counter this problem by artefact relationships via Universally Unique Identifiers (UUIDs) and compliance annotations expressed via dedicated DSLs.

A similar approach to MORSE is described by Amoui and Derakhshanmanesh et al. [8,9,71,72] by using the already mentioned GRAF framework. One part of the framework is a model manager which is responsible for interactions with three repositories: Schema and Constraints, Runtime Model and Model History. The manager offers access to these repositories in form of query and transformation operations. The *Schema and Constraints* repository contains a schema which describes allowed model elements for a runtime model in GRAF. Constraints are declared both at schema level and model level. The *Runtime Model* repository contains a runtime model represented by a graph that describes parts of the adaptable state and a collection of behaviour descriptions. The *Model history* repository stores versioning information of the runtime model to trace back past model transformations.

3.6.6 Overview of approaches

Table 5 gives an overview of approaches according to the architectures introduced in Sect. 3.6: Monolithic, local dataflow, middleware, and repository architectures. Approaches can occur multiple times in the overview if they use more than one of the identified architectures (e.g. a repository in combination with a communication middleware).

Table 5 Overview of approaches related to the identified architectures

Architecture	Related approaches
Monolithic	[17,76,125,193,248–250,277]
Local dataflow	[114,115,135,199]
Model-aware middleware	[5,6,11,14,47,55,57,58,61,66,74,83,91,104–106,107,124,136,194,198,199,217,231,232,234,235,246,255,257,258]
Communication middleware	[14,49,74,88,104,132–134,138,191,199,206,217,226,240,241,279–281]
Repository	[7–9,48,71,72,132–134,191,205,226]

3.7 Actions after runtime model analysis

When models are processed at runtime, different actions are executed as a result of the performed model analysis. In this section, we summarize common actions performed after runtime model analysis such as component replacement, code generation, or manipulation of the analyzed model itself. We discuss *component and connector manipulation, service binding manipulation, management API interaction*, and *code generation* as adaptation and reconfiguration possibilities triggered by requirement and environment changes. Furthermore, we discuss *model manipulation* and *model execution* to update model information for ongoing monitoring and to execute models with operational semantics.

3.7.1 Component and connector manipulation

Component and connector manipulation focuses on adding/removing components or connectors at runtime to adapt the system according to changes in requirements and the environment. It requires a component model which defines the interfaces and elements of components to ensure compatibility. To prove correctness of an adaptation, Inverardi et al. [144] propose a theoretical assume-guarantee framework which analyzes if invariant properties are violated. Their framework can be applied at different levels of abstraction spanning from code to software architecture. For mobile environments, the MADAM project proposes a middleware-based solution to design and implement self-adaptive component-based applications [100]. Variability is obtained by applying a component framework that facilitates dynamic creation of application variants. The component configuration of an application may change at runtime to get the most rewarding variant which is determined by utility functions.

An often used framework to perform such adaptation is OSGi (Open Services Gateway initiative) as dynamic component system for Java where components can be installed, started, stopped, updated, and uninstalled remotely without reboot. A Bundle—the OSGi term for a modularization unit—can be in different states according to its current usage, as depicted in Fig. 15. *Installed* Bundles are successfully loaded, *Resolved* are ready to be started or have stopped, *Starting* ones are being started and wait for activation, *Active* ones are running, *Stopping* ones are being stopped, and *Uninstalled* Bundles have been uninstalled.

In approaches by Alférez and Cetina et al. [3,55–58] the OSGi adaptation mechanism is used in combination with feature models to determine the possible configurations of the system. Morin et al. [180] use OSGi in combination with model comparison between source and target models

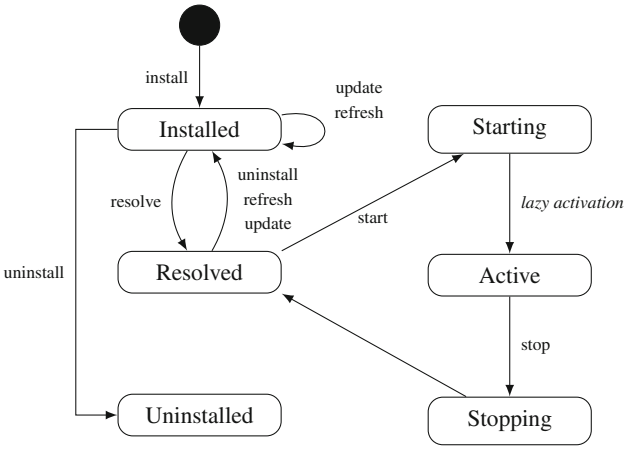

Fig. 15 States of an OSGi Bundle

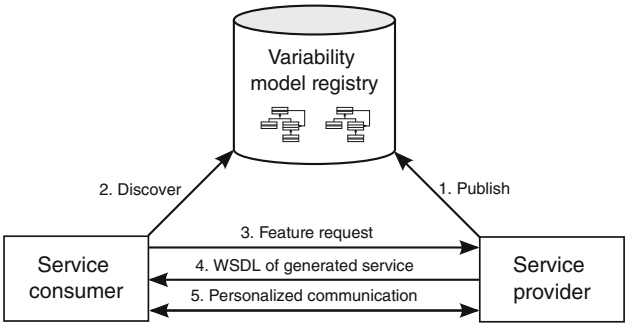

Fig. 16 Simplified view of the service customization approach by Nguyen and Colman [191]

to reason about adaptation steps. Matevska [170] presents an approach called AVORR (Availability Optimized Runtime Reconfiguration) which aims at maintaining continuous availability while maximising service responsiveness during runtime reconfiguration of component-based systems. Welsh et al. [266] use goal models to realize requirements-aware systems by checking assumptions made at design time. If such assumptions do not to hold, system-wide adaptations are applied by swapping components and bindings to enable alternative goal realization strategies. Other approaches use component and connector replacement techniques as well [83–85,126,198,199,201,202,261].

A special approach using runtime models in combination with component-based adaptation is introduced by Thonhauser et al. [248,250]. Applications are defined as an assembly of loosely coupled Model-Based Software Components (MBSCs) which belong to different owners. They cover all relevant domain-specific aspects (like behaviour or data) of a component by using multiple distinct models. Such MBSCs reside in special containers and get executed directly at runtime. Furthermore, these containers have a plugin mechanism to support additional kinds of models.

3.7.2 Service binding manipulation

Similar to component and connector replacement, manipulation of service bindings achieve adaptation through structural change, especially when dealing with QoS requirement restrictions in distributed systems. Cardellini et al. [49] meet QoS requirements in volatile operating environments by using a behavioural model of composite services updated at runtime by a monitoring activity. Changes in the composite service environment lead to a change of bindings between abstract services and the concrete services that implement them. This means that swappable concrete services have to offer the same functionality for a specific abstract service. Such re-arrangement is reasoned from SLAs negotiated with service clients and providers.

Calinescu et al. [45] propose the framework QoSMOS which utilizes Markov models within a feedback loop (see Sect. 3.4.1) to determine quantitatively the reliability and performance quality metrics of service-based systems. The monitoring part of the loop determines performance, reliability and workload of services and updates Markov models accordingly. The results are analyzed and changes are planned in terms of changing the implemented workflow (that is, for example, the service bindings) or modifying resource allocation.

Another form of dynamic service binding is introduced by Nguyen and Colman [191]. They use feature models at runtime to enable web service customization for customers. Figure 16 gives a simplified overview of their approach. First, the service provider publishes a service variability model to a central service registry. The service consumer then fetches the variability model from the registry, selects desired features, and sends the feature compilation request to the provider. The provider composes a service interface consisting of bindings to the requested features and publishes an appropriate WSDL description. The consumer is then able to interact with the personalized service interface.

Grace et al. [116,118] propose the reflective middleware ReMMoC that allows mobile clients to be developed independently of both discovery and interaction mechanisms of web services. Similar interoperability goals are pursued by the frameworks Starlink [39,164] and OverStar [119]. These approaches are described in Sect. 3.6.4.

3.7.3 Management API interaction

After model analysis, required changes can also be applied by calling interface operations which are directly exposed by the system under observation. Such management APIs provided by the system also qualify for extraction of runtime system data in order to apply model-based and control loop techniques on the analyzed behaviour, as described in Sect. 3.4.2.

Possible realizations of system management interfaces are reflection APIs and network-based mechanisms like sockets and services (see Sects. 3.6.2 and 3.6.4, respectively). The already mentioned approach by Huang et al. [136] and Song et al. [231–235] utilizes code generation for wrapping low-level management APIs and uses a system-model adapter to support reading and writing system state in a model-based way, enabling the manipulation of the system at the model level.

3.7.4 Code generation

If model analysis detects adaptation needs and component, service, or management API manipulation are not available, reconfiguration can also be achieved by generating reconfiguration scripts or code which adds additional functionality. Scripts are sent to the controlled system nodes which interpret them to drive the transition from the current to the desired configuration. Because of their dynamic nature, scripting languages like FScript[3] qualify particularly well for writing such reconfiguration scripts, although there is no strict limitation to scripting languages. As an example, in the already mentioned approach by Fleurey and Morin et al. [82,180,181] reconfiguration commands responsible for structural adaptation are instantiated in case of context changes to meet the new requirements.

In case of code generation, Inverardi and Mori [141,142] propose a framework to augment a system with new requirements arising at runtime. At design time, a context model is defined which describes the environment that is beyond the control of the system. The model is updated at runtime to reflect the current environmental situation. When a new requirement is specified, a new system variant is generated. The new variant is loaded by reflection and the entry method invoked to put the new variant in action.

Alternatively, code can also be generated to dynamically enhance interoperability between two systems. Within the CONNECT project,[4] Bencomo et al. [23] introduce an approach where mediators are dynamically generated which translate actions of one networked system to the actions of another networked system developed with no prior knowledge of the former. Runtime models are used to capture meta-information about these systems, including interfaces and additional knowledge about their associated behaviour. Such mediators are also addressed by Hao et al. [130], their approach relies on monitoring for runtime message mismatches and the semi-automatic generation and deployment of behavioural mediators. Mismatch detection is prepared by generated and deployed interceptors which rely on state machines. A monitor then logs the exchange of messages between different systems and supports the detection of message mismatches.

3.7.5 Model manipulation

Direct model manipulation is the common procedure for monitoring approaches to display system information at the model level instead of low-level traces [136,193,217]. Behavioural models enable the possibility to trace system behaviour and workflow activities by extracted system data which is fed directly into the models [76,121,122,133,168,169,275]. Monitoring, simulation and prediction approaches are described in Sect. 3.3.2. Besides monitoring, model manipulation is also used for incremental refinement of the model data, e.g. when using performance models to reflect the current timing constraints given by contextual conditions [47,48,104]. Such refinement of model data can be used for self-healing mechanisms where the system is able to update its own model dependent on the current situation.

3.7.6 Model execution

Another possible action after runtime model analysis is the direct execution of the model. As described throughout Sect. 3.4.6, model execution deals with direct execution of models containing operational semantics. A popular example of executable models are state machines which are used throughout our identified literature [11,17,76,78,80,103,125,135,156,167,177,186,206,248,249,273,275,279–281].

3.7.7 Overview of approaches

Table 6 gives an overview of approaches according to the actions introduced in Sect. 3.7: Component and connector manipulation, service binding manipulation, management

Table 6 Overview of approaches related to the identified actions

Action	Related approaches
Component and connector manipulation	[3,47,48,55–58,83–85,100,126,170,180,198,199,201,205,206,246,248,250,261,266,279–281]
Service binding manipulation	[39,45,49,55,56,116,118,119,164,191]
Management API interaction	[20,91,95,96,105–107,136,231–235,255,257,258,261]
Code generation	[23,82,85,124,141,142,180–182,198,199]
Model manipulation	[47,48,76,104,121,122,133,136,168,169,193,217]
Model execution	[8,9,11,17,38,71,72,76,78,80,88,90,103,125,135,156,167,177,186,195,206,226,237,248,249,273,275,277,279–281]

[3] See: http://fractal.ow2.org/fscript/.

[4] See: https://www.connect-forever.eu/.

API interaction, code generation, model manipulation, and model execution.

3.8 Research methods and empirical evidence of runtime model approaches

While extracting objectives, techniques, architectures, and kinds of models, we also analyzed research methods applied in literature to evaluate the presented approaches as well as the resulting empirical evidence. The research methods are categorized into five different groups [75]:

- Controlled experiment
- Case study
- Survey research
- Ethnography
- Action research

A *controlled experiment* is "an investigation of a testable hypothesis where one or more independent variables are manipulated to measure their effect on one or more dependent variables" [75]. An example would be the measurement of productivity with and without the usage of a specific software.

We define *case study* as a practical demonstration of an introduced approach with a variable depth of industrial relevance. Case studies can be either exploratory (observe some phenomena to obtain new knowledge) or confirmatory (test approach against existing theories). In contrast to survey research, case studies usually do not have representative sampling.

Survey research contains the "selection of a representative sample from a well-defined population, and the data analysis techniques used to generalize from that sample to the population" [75]. An example of such a survey research would be an analysis of the behaviour of selected developers to derive information about developer behaviour in general.

Ethnography is a research method where social interactions between people of a specific community are examined through field observation, especially to gain knowledge about communication culture within that community [208]. The researcher can either observe the community from an external point of view or join the community to gain insight into the community behaviour.

Action research concentrates on solving problems while observing the process of problem solving itself. Researchers reflect over past, current, and planned actions to evaluate their impacts on the problem solving process in terms of efficiency and level of improvement.

In our research, only controlled experiments and case studies were conducted in the identified literature, other forms of research methods listed above were not applied (at least not according to the research method definitions given by Easterbrook et al. [75]). Our research yielded 10 controlled experiments and 117 case studies within our literature search results, whereas 37 case studies are applied in realistic and 80 in exemplary test scenarios.

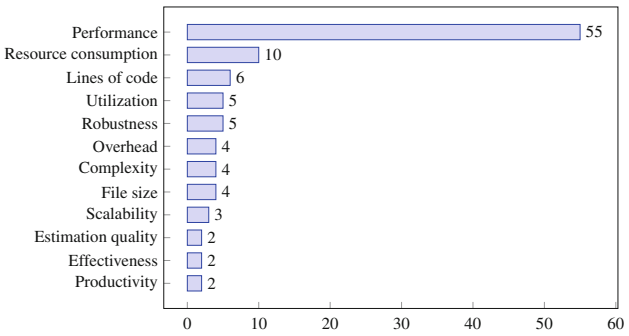

Fig. 17 Absolute frequency of metrics used to evaluate runtime model approaches

In addition to the research methods, we took note of metrics used to evaluate the identified approaches, e.g. performance measurements, lines of code, resource consumption, and model complexity. Figure 17 gives an overview of identified metrics regarding absolute frequency of their usage. *Performance* covers metrics like response/synchronization/execution time, latency, down time, reboot delay, and transactions per second. *Resource consumption* encompasses metrics like memory usage and amount of disk I/O operations. *Lines of code* measurements involve code sizes when comparing different approaches or implementation strategies. *Utilization* deals with resource occupation measurements, e.g. the utilization of network routers before and after reconfiguration [47,48] or node utilization in case of virtual machine redeployment [137]. Evaluation regarding the number of model elements, components, instances and object links are summarized in metric *complexity*. *Overhead* measures the ratio of intervention efforts (e.g. reconfiguration/monitoring actions or execution of instrumentation instructions) in comparison with the regular workflow. The *file size* metric describes the sizes of models stored in files or components enriched with embedded data. *Robustness* deals with measurements of fault tolerance, error distribution and anomaly detection rates. *Scalability* measurements encompass analysis of system behaviour under various workloads as well as stress test scenarios. Comparisons of estimated parameters with real data measurements are summarized in metric *estimation quality*. The *effectiveness* metric covers the ratio of successful versus unsuccessful actions, e.g. the comparison of successful and unsuccessful reconfiguration actions or preventions of SQL injection attacks [125]. *Productivity* deals with productivity improvements gained through usage of the respective approach in a realistic, business relevant test scenario. As Fig. 17 outlines, performance measurement is by far the most often used method for evaluating approaches dealing with models at runtime.

Table 7 Overview of approaches related to the applied research methods

Research method	Related approaches
Controlled experiment	[7,19,47,48,52,92,95,126,137,171]
Case study	*Realistic scenario*: [2,8,9,14,18,23,31,54,71,72,112,123–126,128,131,133,138,157,180,187,198,199,211,223,224,226,229,233] [240,243,248,250,265,272,278] *Exemplary scenario*: [3,11,15,32,35–37,49,55,57,58,61,62,77,79,81–85,88,90,91,95,96,98,99,101,104,115,118,119,125,126,135,136] [142,146,149,167,169,170,176,185,189,191,203–206,217,219,220,222,231,232,234,235,237,244,246] [248,249,253–255,258,259,261,262,266–268,271,276,277,279–281,283]
Survey research	None
Ethnography	None
Action research	None

Table 7 gives an overview of approaches according to the research methods described above. As mentioned, none of the identified literature uses survey research, ethnography or action research to evaluate the presented approaches.

3.9 Summary and survey publications

A number of publications provide summaries or surveys related to models at runtime, which we summarize below. None of those publications provides a systematic literature review.

Bencomo [20] points out the importance of using models at runtime. The author argues that long-living software systems need first-class representations of themselves to support dynamic changes. Furthermore, the need of a more precise distinction between development models and runtime models is demanded, which our approach partially assists by underlining various kinds of runtime models. One open research question of the paper addresses the problem of synchronization between runtime models and the running system. We help answering this question by our summary of techniques for processing runtime models: Introspection, conformance, comparison, transformation, and execution.

Blair et al. [34] also point out the important role of software models at runtime. They present relevant articles and emphasize several research challenges regarding models at runtime. They divide models into several groups: Structure versus behaviour, procedural versus declarative, functional versus non-functional, and formal versus informal. While not grouping our identified kinds of models according to these classes, in contrast to their classification our work focuses on objectives, techniques, and architectures when using models at runtime.

Vogel et al. [256,259] list different classes of models used at runtime: *Implementation models* are platform-specific and coupled to the system's implementation, *Configuration and Architectural Models* reflect the current system configuration and provide architectural views, *Context and Resource Models* describe the environment of a system and *Configuration Space and Variability Models* reveal possible variants of a system. Their approach focus on the relations between these models while our work emphasizes objectives, techniques, and architectures of runtime models in general.

As mentioned in Sect. 3.3.1, adaptation is one main objective to cope with continuously changing requirements. Our literature search yielded proceedings of conferences specializing in adaptive and self-managing systems [213], international workshops on models in general [165] and workshops on ubiquitous systems [216]. All these topics relate to adaptation in general, giving us confidence that the importance of runtime models will increase in the future. As seen in the introduced approaches throughout the article, runtime models are an effective instrument to tackle various problems regarding adaptation needs such as requirement evolution and changing environments.

4 Discussion

In this section, we assess our literature search method, the precision of our search results, possible distortions by our selection criteria and limitations of this study. Furthermore, we discuss our collected results and relate them to our research questions introduced in Sect. 2.2. At the end of this section, we discuss perspectives and future challenges of models at runtime.

4.1 Search method

In each SLR, the question arises whether relevant publications were omitted due to our literature search method. For instance, we could have improved our search results by including more publishers. This would potentially have led to an increased number of publications but would also complicate the phases of filtering, duplicate elimination, and grouping. We decided to choose only the ACM Guide to Computing Literature and IEEE Xplore, which include bibliographic information from all major publishers in computing (e.g. Springer, Elsevier, etc.), and compensate possibly missed publications by extending our search to other publishers by our snowballing phase. The low number of addi-

tional publications found during snowballing indicates that our search has found most of the relevant literature.

Another possible distortion in our search results may result from publication filtering by looking at their abstracts. Abstracts can be very short or even not be available, bringing inaccuracy into our search. Furthermore, we realized that some of the abstracts suffer from incorrect formatting (e.g. words of a sentence sticking together). We are unsure whether this is just a presentation issue or whether this has bad influence on the search functionality. We compensate such perturbations by including multiple spellings of words and abbreviations in our search string and by shifting inclusion/exclusion decisions to later, more detailed phases in case of uncertainty. Again, the low number of additional publications found during snowballing indicates that our strategy worked well.

We applied our snowballing phase only to papers to be discussed in detail. While we could apply snowballing to other publications as well, we decided to keep the number of additional literature manageable, expecting that references of non-detailed papers would again yield papers not fitting the topic exactly. Keeping the number of papers manageable was also achieved by dismissing publications if uncertainty about relevance and classification persisted until the elimination phase.

Overall, despite a structured proceeding, a minimal degree of subjectivity regarding decisions about inclusion and exclusion of publications remains, especially when weighing fitting/not fitting the topic, importance/unimportance of facts and classification of literature.

4.2 Classification into objectives, techniques, architectures, and kinds of models

In this article, we analyze objectives, techniques, architectures, and kinds of models when using models at runtime to identify research areas not covered so far and to provide a comprehensive overview for researchers who are new to this topic. Our classification contains some overlaps because there exist interconnections between the aspects (e.g. (model-)conformance is considered as objective and technique). Figure 18 shows the relationships between the four aspects.

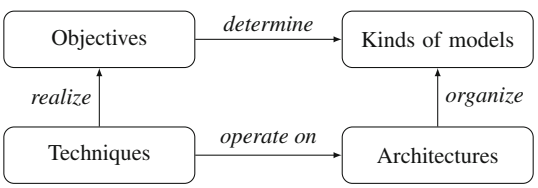

Fig. 18 Interconnections of the proposed classification aspects

Objectives determine the kinds of models which can be used to support the satisfaction of these objectives (e.g. the objective *error handling* can be satisfied by using state machine models, but hardly by using feature models). Architectures organize the selected kinds of models and make them available at runtime through various interfaces (e.g. a *repository architecture* enables access to business process models through a special query interface). Techniques are responsible for realizing objectives by utilizing model access interfaces provided by the architecture (e.g. within a monolithic architecture, the technique *model comparison* gets the architecture model from an XMI file, compares it with the current configuration, and calculates corrective actions, thus satisfying the objective of adaptation).

In other words, for every objective there exists a set of kinds of models which help to satisfy that objective when used at runtime (see Table 4 for details). Architectures and techniques are independent of the pursued objective and the used kinds of models (an exception here is the technique *model execution* since not every kind of model is executable; furthermore, it is debatable whether this technique is related to the topic of models at runtime—see Sect. 3.4.6).

Our proposed classification is the only one possibility to structure runtime model approaches, and other classifications also qualify for giving an overview of various runtime model aspects (e.g. different model query techniques/languages, synchronization mechanisms and feedback loop variations).

4.3 Research questions

In Sect. 2.2, we introduced four different research questions and were able to answer these the following way according to our research results:

For what purposes are models used at runtime? We answer this by identifying different objectives throughout our literature analysis: Adaptation, abstraction, consistency, conformance, error handling, monitoring, simulation, prediction, platform independence and policy checking/enforcement. A common objective of all these classes is the shift from low-level system interactions to model-based processing which is closer to the problem space. Recent research introduces requirements-based and goal-oriented mechanisms to continue this shift further in the problem space direction [53,215,260].

Which techniques are applied when processing models at runtime? We identified the following techniques when processing models at runtime: Autonomic control loops, introspection, model conformance, model comparison, model transformation, and model execution. These techniques address the ongoing surveillance of a running system, the extraction of its data, and the model-based processing to reason about the system and to affect its execution. As described in Sect. 3.4.6, there is still no consensus about the

relationship between model execution and the models at runtime paradigm.

Which problems are addressed by using models at runtime? Our classification of problems includes inaccurate predictions, changing environments and requirements, combinatorial explosion of system variants, error localization, rule enforcement and hand-written artefacts with low reusability, maintenance and abstraction. Early approaches focussed on architectural adaptation to tackle the problems of changing environments and requirements. Over time researches realized the potential of models at runtime and extended their field of application to cope with additional problems. Ongoing research on models at runtime indicates that there will be more extensions in the near future.

Which architectures exist for processing models at runtime? Identified approaches use monolithic, local dataflow, middleware and repository architectures which often overlap in their usage. Such architectures must include introspective capabilities to extract runtime information and to relate the collected data with models. Models at runtime differ from reflection mainly in terms of abstraction: Reflection is solution space-oriented, while models at runtime operate on a higher level of abstraction towards problem space [34]. The selection of an architecture which supports runtime models could be assisted by using architectural decision templates [282] (see Sect. 4.4 for details).

Which research methods are most frequently used for evaluating runtime model approaches? Which empirical evidence has been reported? Evaluation and corresponding evidence for models at runtime is most often provided through example scenarios only or not at all. In only 37 cases, case studies have been conducted within realistic settings. Only 10 controlled experiments have been conducted. Often metrics are used for evaluation. As Fig. 17 outlines, performance measurement is by far the most often used method for evaluating approaches dealing with models at runtime.

Besides purposes, techniques, problems, and architectures, we identified different kinds of models used in our selected literature. We pointed out the broad field of meanings, use cases and representations of runtime models. We did not distinguish between different model representations (e.g. graphical, textual) to capture all publications using the term *model* in combination with runtime utilization.

As demonstrated by our presentation of the most relevant approaches, models used at runtime cover many different research areas leading from low-level adaptation on AST-level to High Performance Computing. The majority of our analyzed literature deals with model-based monitoring and software adaptation according to changing environments and requirements. Few approaches address trace visualization and parameter prediction. Other areas not addressed in detail so far are model-based monitoring, adaptation, and architecture reconstruction of software where introspection mechanisms - namely event log checking, instrumentation and management APIs — cannot be applied easily, as is the case with legacy systems.

4.4 Perspectives and future challenges

Early approaches using models at runtime focussed on architecture-based mechanisms to realize monitoring and adaptation [95,96,98]. The ideas further evolved from early middleware-based adaptation approaches [63,65,74,152,210] to other topics like error handling [135,145] and model execution (see Sect. 3.4.6). Several approaches refined the management of causal connections between models and the running system [105–108,136,231–235,255,257,258]. Recent approaches refine established techniques like feedback loops [59,155,157,202] and abstract running systems even further by focussing on goal- and requirement-oriented aspects [53,215,260]. Overall, there has been a recognizable shift from simple adaptation interests to wider application fields and more goal-oriented and user-centric approaches over the last years [30]. The question arises in which direction models at runtime will evolve in the future.

While researchers put a lot of effort into the development of monitoring, adaptation, and synchronization mechanisms, we think that there is a strong need for process-related enhancements when designing and implementing systems which are augmented by runtime models. Differences to traditional development processes have to be elaborated, especially in terms of testing to ensure quality of the resulting application. The vast amount of models combined with different architectures and techniques lead to a great number of test cases which have to be executed to reduce the possibility of runtime errors in such highly dynamic environments. We believe that a tool-assisted selection of architectures, techniques, kinds of models, and testing mechanisms would boost the development and adoption of applications using models at runtime. Reusable architectural decision templates would be a first step to realize such an assisting framework [282].

5 Conclusions

In this article, we applied a systematic literature review to analyze objectives, techniques, kinds, and architectures when using models at runtime. Our search yielded a classification of distinct groups containing 122 papers analyzed in detail, 87 not detailed, and 33 dismissed papers. We identified different objectives throughout our literature analysis, namely adaptation, abstraction, consistency, conformance, error handling, monitoring, simulation, prediction, platform independence, and policy checking/enforcement. We demonstrated the usage of different kinds of models in our identified literature to achieve these objectives and to tackle common

problems of inaccurate predictions, changing environments, error localization, hand-written artefacts, and combinatorial explosions regarding system configurations. We identified different techniques when processing runtime models to extract runtime data, raise the abstraction level and enforce constraints: Introspection, model conformance, model comparison, model transformation, and model execution. We demonstrated the usage of these techniques in our selected literature and explained their similarities and differences. Regarding architectures, identified approaches use monolithic, local dataflow, middleware, and repository architectures which often overlap in their usage.

With this work, we aim to provide a basis for future studies and to promote ideas for research areas not covered in detail by models at runtime so far. In particular, we aim to automate parts of the runtime model development process and improve the architectural decision process when designing systems with runtime model support.

References

1. Ahmad, M.: First step towards a domain specific language for self-adaptive systems. In: New Technologies of Distributed Systems (NOTERE), 2010 10th Annual International Conference on, pp. 285–290 (2010)
2. Ahn, Y., Yeo, I., Bettati, R.: Efficient calibration of thermal models based on application behavior. In: Proceedings of the 2009 IEEE International Conference on Computer Design ICCD'09, pp. 41–46. IEEE Press, Piscataway (2009)
3. Alferez, G.H., Pelechano, V.: Context-aware autonomous web services in software product lines. In: Proceedings of the 2011 15th International Software Product Line Conference SPLC '11, pp. 100–109. IEEE Computer Society, Washington, DC, USA (2011)
4. Alférez, G.H., Pelechano, V.: Dynamic evolution of context-aware systems with models at runtime. In: Proceedings of the 15th International Conference on Model Driven Engineering Languages and Systems MODELS'12, pp. 70–86. Springer, Berlin (2012)
5. Alia, M., Eliassen, F., Hallsteinsen, S., Stav, E.: Madam: towards a flexible planning-based middleware. In: Proceedings of the 2006 International Workshop on Self-Adaptation and Self-Managing Systems SEAMS '06, pp. 96–96. ACM, New York (2006)
6. Alia, M., Horn, G., Eliassen, F., Khan, M., Fricke, R., Reichle, R.: A component-based planning framework for adaptive systems. In: Meersman, R., Tari, Z. (eds.) On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE volume 4276 of Lecture Notes in Computer Science, pp. 1686–1704. Springer, Berlin (2006)
7. Almorsy, M., Grundy, J., Ibrahim, A.S.: Mdse@r: model-driven security engineering at runtime. In: Proceedings of the 4th International Conference on Cyberspace Safety and Security CSS'12, pp. 279–295. Springer, Berlin (2012)
8. Amoui, M., Derakhshanmanesh, M., Ebert, J., Tahvildari, L.: Software evolution towards model-centric runtime adaptivity. In: Proceedings of the 2011 15th European Conference on Software Maintenance and Reengineering CSMR '11, pp. 89–92. IEEE Computer Society, Washington, DC, USA (2011)
9. Amoui, M., Derakhshanmanesh, M., Ebert, J., Tahvildari, L.: Achieving dynamic adaptation via management and interpretation of runtime models. J. Syst. Softw. **85**(12), 2720–2737 (2012)
10. Andersen, A., Blair, G., Coulson, G.: The design and implementation of openorb v2. page n/a (2001)
11. Arcaini, P., Gargantini, A., Riccobene, E.: Coma: conformance monitoring of java programs by abstract state machines. In: Proceedings of the Second International Conference on Runtime Verification RV'11, pp. 223–238. Springer, Berlin (2012)
12. Ardagna, D., Ghezzi, C., Mirandola, R.: Rethinking the use of models in software architecture. In: Proceedings of the 4th International Conference on Quality of Software-Architectures: Models and Architectures QoSA '08, pp. 1–27. Springer, Berlin (2008)
13. Aßmann, U., Bencomo, N., Cheng, B.H.C., France, R.B.: Models@run.time (dagstuhl seminar 11481). Dagstuhl Rep. **1**(11), 91–123 (2011)
14. Avouac, P.-A., Lalanda, P., Nigay, L.: Autonomic management of multimodal interaction: dynamo in action. In: Proceedings of the 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems EICS '12, pp. 35–44. ACM, New York (2012)
15. Bai, X., Liu, Y., Wang, L., Tsai, W.-T., Zhong, P.: Model-based monitoring and policy enforcement of services. In: Proceedings of the 2009 Congress on Services—I Services '09, pp. 789–796. IEEE Computer Society, Washington, DC, USA (2009)
16. Barbier, F.: Mde-based design and implementation of autonomic software components. In: Cognitive Informatics, 2006. ICCI 2006. 5th IEEE International Conference on vol. 1, pp. 163–169 (2006)
17. Barbier, F., Ballagny, C.: Proved metamodels as backbone for software adaptation. In: Proceedings of the 2010 IEEE 12th International Symposium on High-Assurance Systems Engineering, HASE '10, pp. 114–121. IEEE Computer Society, Washington, DC, USA (2010)
18. Bauer, A., Jürjens, J., Yu, Y.: Run-time security traceability for evolving systems and &#8224. Comput. J. **54**(1), 58–87 (2011)
19. Beltrame, G., Sciuto, D., Silvano, C., Lyonnard, D., Pilkington, C.: Exploiting tlm and object introspection for system-level simulation. In: Proceedings of the Conference on Design, Automation and Test in Europe: Proceedings, DATE '06, pp. 100–105, 3001 Leuven, Belgium, Belgium, European Design and Automation Association (2006)
20. Bencomo, N.: On the use of software models during software execution. In: Proceedings of the 2009 ICSE Workshop on Modeling in Software Engineering MISE '09, pp. 62–67. Washington, DC, USA, IEEE Computer Society (2009)
21. Bencomo, N., Belaggoun, A.: Supporting decision-making for self-adaptive systems: from goal models to dynamic decision networks. In: Doerr, J., Opdahl, A. (eds). Requirements Engineering: Foundation for Software Quality vol. 7830 of Lecture Notes in Computer Science (pp. 221–236). Springer, Berlin (2013)
22. Bencomo, N., Belaggoun, A., Issarny, V.: Dynamic decision networks for decision-making in self-adaptive systems: a case study. In: Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems SEAMS '13, pp. 113–122. IEEE Press, Piscataway (2013)
23. Bencomo, N., Bennaceur, A., Grace, P., Blair, G., Issarny, V.: The role of models@run.time in supporting on-the-fly interoperability. Computing **95**(3), 167–190 (2013)
24. Bencomo, N., Blair, G., France, R.: Summary of the workshop models@run.time at models 2006. In: Proceedings of the 2006 International Conference on Models in Software Engineering, MoDELS'06, pp. 227–231. Springer, Berlin (2006)
25. Bencomo, N., Blair, G., France, R., Muñoz, F., Jeanneret, C.: Models in software engineering. In: Third International Workshop on Models@run.time, pp. 90–96. Springer, Berlin (2009)
26. Bencomo, N., Blair, G., France, R., Muñoz, F., Jeanneret, C.: 4th international workshop on models@run.time. In: Proceedings of the 2009 International Conference on Models in Software Engineering MODELS'09, pp. 119–123. Springer, Berlin (2010)

27. Bencomo, N., Blair, G., Götz, S., Morin, B., Rumpe, B.: Report on the 7th international workshop on models@run.time. SIGSOFT Softw. Eng. Notes **38**(1), 27–30 (2013)
28. Bencomo, N., Blair, G.S., France, R.B.: Models@run.time workshops. http://www.comp.lancs.ac.uk/bencomo/MRT/. Accessed: 28/12/2012
29. Bencomo, N., France, R., Blair, G.: Models in software engineering. In: Second International Workshop on Models@run.time, pp. 206–211. Springer, Berlin (2008)
30. Bencomo, N., Whittle, J., Sawyer, P., Finkelstein, A., Letier, E.: Requirements reflection: requirements as runtime entities. In: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering—vol. 2 ICSE '10, pp. 199–202. ACM, New York (2010)
31. Bertolino, A., Calabrò, A., Lonetti, F., Di Marco, A., Sabetta, A.: Towards a model-driven infrastructure for runtime monitoring. In: Proceedings of the Third International Conference on Software Engineering for Resilient Systems SERENE'11, pp. 130–144. Springer, Berlin (2011)
32. Betermieux, S., Bomsdorf, B.: Finalizing dialog models at runtime. In: Proceedings of the 7th International Conference on Web Engineering ICWE'07, pp. 137–151. Springer, Berlin (2007)
33. Beugnard, A., Jézéquel, J.-M., Plouzeau, N., Watkins, D.: Making components contract aware. Computer **32**(7), 38–45 (1999)
34. Blair, G., Bencomo, N., France, R.B.: Models@run.time. Computer **42**(10), 22–27 (2009)
35. Blumendorf, M., Feuerstack, S., Albayrak, S.: Multimodal user interfaces for smart environments: the multi-access service platform. In: Proceedings of the Working Conference on Advanced Visual Interfaces AVI '08, pp. 478–479. ACM, New York (2008)
36. Blumendorf, M., Lehmann, G., Feuerstack, S., Albayrak, S.: Interactive systems. design, specification, and verification. In: Executable Models for Human-Computer Interaction, pp. 238–251. Springer, Berlin (2008)
37. Bodenstaff, L., Wombacher, A., Reichert, M., Wieringa, R.: Made4ic: an abstract method for managing model dependencies in inter-organizational cooperations. Serv. Oriented Comput. Appl. **4**(3), 203–228 (2010)
38. Bomsdorf, B., Grau, S., Hudasch, M., Milde, J.-T.: Configurable executable task models supporting the transition from design time to runtime. In: Proceedings of the 14th International Conference on Human-Computer Interaction: Design and Development Approaches—Volume Part I, HCII'11, pp. 155–164. Springer, Berlin (2011)
39. Bromberg, Y.-D., Grace, P., Réveillère, L.: Starlink: Runtime interoperability between heterogeneous middleware protocols. In: Proceedings of the 2011 31st International Conference on Distributed Computing Systems ICDCS '11, pp. 446–455. IEEE Computer Society, Washington, DC, USA (2011)
40. Bruno, G., La Rosa, M.: From collaboration models to bpel processes through service models. In: Proceedings of the Third International Conference on Business Process Management BPM'05, pp. 75–88. Springer, Berlin (2006)
41. Buchheit, M., Klein, R., Nutt, W.: Constructive Problem Solving: A Model Construction Approach Towards Configuration. Technical report, DFKI (1995)
42. Budgen, D., Burn, A.J., Brereton, O.P., Kitchenham, B.A., Pretorius, R.: Empirical evidence about the uml: a systematic literature review. Softw. Pract. Exper. **41**(4), 363–392 (2011)
43. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M.: Pattern-Oriented Software Architecture: A System of Patterns, Vol. 1. Number Bd. 1. Wiley India Pvt. Limited (2008)
44. Calinescu, R., France, R., Ghezzi, C.: Editorial. Computing **95**(3), 165–166 (2013)
45. Calinescu, R., Grunske, L., Kwiatkowska, M., Mirandola, R., Tamburrelli, G.: Dynamic qos management and optimization in service-based systems. IEEE Trans. Softw. Eng. **37**(3), 387–409 (2011)
46. Callow, G., Watson, G., Kalawsky, R.: System modelling for runtime verification and validation of autonomous systems. In: System of Systems Engineering (SoSE), 2010 5th International Conference on, pp. 1–7 (2010)
47. Caporuscio, M., Di Marco, A., Inverardi, P.: Model-based system reconfiguration for dynamic performance management. J. Syst. Softw. **80**(4), 455–473 (2007)
48. Caporuscio, M., Marco, A.D., Inverardi, P.: Run-time performance management of the siena publish/subscribe middleware. In: Proceedings of the 5th International Workshop on Software and Performance, WOSP '05, pp. 65–74. ACM, New York (2005)
49. Cardellini, V., Casalicchio, E., Grassi, V., Lo Presti, F., Mirandola, R.: Qos-driven runtime adaptation of service oriented architectures. In: Proceedings of the the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering, ESEC/FSE '09, pp. 131–140. ACM, New York (2009)
50. Cariou, E., Barbier, F., Goaer, O.L.: Model execution adaptation? In: Proceedings of the 7th Workshop on Models@run.time MRT '12, pp. 60–65. ACM, New York (2012)
51. Casale, G., Harrison, P.: A class of tractable models for run-time performance evaluation. In: Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering ICPE '12, pp. 63–74. ACM, New York (2012)
52. Casanova, P., Schmerl, B., Garlan, D., Abreu, R.: Architecture-based run-time fault diagnosis. In: Proceedings of the 5th European Conference on Software Architecture, ECSA'11, pp. 261–277. Springer, Berlin (2011)
53. Cavallaro, L., Sawyer, P., Sykes, D., Bencomo, N., Issarny, V.: Satisfying requirements for pervasive service compositions. In: Proceedings of the 7th Workshop on Models@run.time MRT '12, pp. 17–22. ACM, New York (2012)
54. Cazzola, W., Ghoneim, A., Saake, G.: Software evolution through dynamic adaptation of its oo design. In: Ryan, M., Meyer, J.-J., Ehrich, H.-D. (eds.) Objects, Agents, and Features, volume 2975 of Lecture Notes in Computer Science, pp. 67–80. Springer, Berlin (2004)
55. Cetina, C., Fons, J., Pelechano, V.: Applying software product lines to build autonomic pervasive systems. In: Proceedings of the 2008 12th International Software Product Line Conference SPLC '08, pp. 117–126. IEEE Computer Society, Washington, DC, USA (2008)
56. Cetina, C., Giner, P., Fons, J., Pelechano, V.: A model-driven approach for developing self-adaptive pervasive systems
57. 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), 37–43 (2009)
58. Cetina, C., Giner, P., Fons, J., Pelechano, V.: Using feature models for developing self-configuring smart homes. In: Proceedings of the 2009 Fifth International Conference on Autonomic and Autonomous Systems ICAS '09, pp. 179–188. IEEE Computer Society, Washington, Dc, USA (2009)
59. Chaari, T., Fakhfakh, K.: Semantic modeling and reasoning at runtime for autonomous systems engineering. In: Proceedings of the 2012 9th International Conference on Ubiquitous Intelligence and Computing and 9th International Conference on Autonomic and Trusted Computing UIC-ATC '12, pp. 415–422. IEEE Computer Society, Washington, DC, USA (2012)
60. Chang, H., Collet, P.: Compositional patterns of non-functional properties for contract negotiation. JSW **2**(2), 52–63 (2007)
61. Chen, D., Törngren, M., Persson, M., Feng, L., Qureshi, T.N.: Towards model-based engineering of self-configuring embedded systems. In: Proceedings of the 2007 International Dagstuhl Con-

ference on Model-Based Engineering of Embedded Real-Time Systems, MBEERTS'07, pp. 345–353. Springer, Berlin (2010)
62. Cheng, S.-W., Garlan, D., Schmerl, B.R., Sousa, J.A.P., Spitznagel, B., Steenkiste, P., Hu, N.: Software architecture-based adaptation for pervasive systems. In: Proceedings of the International Conference on Architecture of Computing Systems: Trends in Network and Pervasive Computing ARCS '02, pp. 67–82. Springer, London (2002)
63. Costa, F., Provensi, L.L., Vaz, F.F.: Towards a more effective coupling of reflection and runtime metamodels for middleware. In: Workshop on Models at Runtime (2006)
64. Costa, F.M., Blair, G.S.: Integrating meta-information management and reflection in middleware. In: Proceedings of the International Symposium on Distributed Objects and Applications DOA '00, p. 133. IEEE Computer Society, Washington, DC, USA (2000)
65. Coulson, G., Blair, G., Grace, P., Taiani, F., Joolia, A., Lee, K., Ueyama, J., Sivaharan, T.: A generic component model for building systems software. ACM Trans. Comput. Syst. **26**(1), 1:1–1:42 (2008)
66. Coutaz, J., Balme, L., Alvaro, X., Calvary, G., Demeure, A., Sottet, J.-S.: An mde-soa approach to support plastic user interfaces in ambient spaces. In: Proceedings of the 4th International Conference on Universal access in Human-Computer Interaction: Ambient interaction UAHCI'07, pp. 63–72. Springer, Berlin (2007)
67. Criado, J., Iribarne, L., Padilla, N., Troya, J., Vallecillo, A.: An mde approach for runtime monitoring and adapting component-based systems: application to wimp user interface architectures. In: Proceedings of the 2012 38th Euromicro Conference on Software Engineering and Advanced Applications SEAA '12, pp. 150–157. IEEE Computer Society, Washington, DC, USA (2012)
68. Dalpiaz, F., Serral, E., Valderas, P., Giorgini, P., Pelechano, V.: A nfr-based framework for user-centered adaptation. In: Proceedings of the 31st International Conference on Conceptual Modeling ER'12, pp. 439–448. Springer, Berlin (2012)
69. Daubert, E., Fouquet, F., Barais, O., Nain, G., Sunye, G., Jezequel, J.-M., Pazat, J.-L., Morin, B.: A models@runtime framework for designing and managing service-based applications. In: Software Services and Systems Research—Results and Challenges (S-Cube), 2012 Workshop on European, pp. 10–11 (2012)
70. de Roo, A., Sozer, H., Aksit, M.: Runtime verification of domain-specific models of physical characteristics in control software. In: Secure Software Integration and Reliability Improvement (SSIRI), 2011 Fifth International Conference on pp. 41–50 (2011)
71. Derakhshanmanesh, M., Amoui, M., O'Grady, G., Ebert, J., Tahvildari, L.: Graf: graph-based runtime adaptation framework. In: Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems SEAMS '11, pp. 128–137. ACM, New York (2011)
72. Derakhshanmanesh, M., Salehie, M., Ebert, J.: Towards model-centric engineering of a dynamic access control product line. In: Proceedings of the 16th International Software Product Line Conference—Volume 2 SPLC '12, pp. 151–155. ACM, New York (2012)
73. Dobson, S., Denazis, S., Fernández, A., Gaïti, D., Gelenbe, E., Massacci, F., Nixon, P., Saffre, F., Schmidt, N., Zambonelli, F.: A survey of autonomic communications. ACM Trans. Auton. Adapt. Syst. **1**(2), 223–259 (2006)
74. Dubus, J., Merle, P.: Applying omg d & c specification and eca rules for autonomous distributed component-based systems. In: Proceedings of the 2006 International Conference on Models in Software Engineering MoDELS'06, pp. 242–251. Springer, Berlin (2006)
75. Easterbrook, S., Singer, J., Storey, M.A., Damian, D.: Selecting empirical methods for software engineering research
76. Elkorobarrutia, X., Muxika, M., Sagardui, G., Barbier, F., Aretxandieta, X.: A framework for statechart based component reconfiguration. In: Proceedings of the Fifth IEEE Workshop on Engineering of Autonomic and Autonomous Systems EASE '08, pp. 37–45. IEEE Computer Society, Washington, DC, USA (2008)
77. Epifani, I., Ghezzi, C., Mirandola, R., Tamburrelli, G.: Model evolution by run-time parameter adaptation. In: Proceedings of the 31st International Conference on Software Engineering ICSE '09, pp. 111–121. IEEE Computer Society, Washington, DC, USA (2009)
78. Feuerstack, S., Anjo, M.D.S., Pizzolato, E.B.: Model-based design and generation of a gesture-based user interface navigation control. In: Proceedings of the 10th Brazilian Symposium on on Human Factors in Computing Systems and the 5th Latin American Conference on Human-Computer Interaction IHC+CLIHC '11, pp. 227–231, Porto Alegre, Brazil, Brazil, Brazilian Computer Society (2011)
79. Feuerstack, S., Blumendorf, M., Schwartze, V., Albayrak, S.: Model-based layout generation. In: Proceedings of the Working Conference on Advanced Visual Interfaces AVI '08, pp. 217–224. ACM, New York (2008)
80. Feuerstack, S., Pizzolato, E.B.: Engineering device-spanning, multimodal web applications using a model-based design approach. In: Proceedings of the 18th Brazilian Symposium on Multimedia and the web, WebMedia '12, pp. 29–38.ACM, New York (2012)
81. Filieri, A., Ghezzi, C., Tamburrelli, G.: Run-time efficient probabilistic model checking. In: Proceedings of the 33rd International Conference on Software Engineering ICSE '11, pp. 341–350. ACM, New York (2011)
82. Fleurey, F., Dehlen, V., Bencomo, N., Morin, B., Jézéquel, J.-M.: Models in software engineering. In: Modeling and Validating Dynamic Adaptation, pp. 97–108. Springer, Berlin (2009)
83. Floch, J., Hallsteinsen, S., Stav, E., Eliassen, F., Lund, K., Gjorven, E.: Using architecture models for runtime adaptability. IEEE Softw. **23**(2), 62–70 (2006)
84. Fouquet, F., Daubert, E., Plouzeau, N., Barais, O., Bourcier, J., Jézéquel, J.-M.: Dissemination of reconfiguration policies on mesh networks. In: Proceedings of the 12th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems DAIS'12, pp. 16–30. Springer, Berlin (2012)
85. Fouquet, F., Morin, B., Fleurey, F., Barais, O., Plouzeau, N., Jezequel, J.-M.: A dynamic component model for cyber physical systems. In: Proceedings of the 15th ACM SIGSOFT Symposium on Component Based Software Engineering CBSE '12, pp. 135–144. ACM, New York (2012)
86. Fouquet, F., Nain, G., Morin, B., Daubert, E., Barais, O., Plouzeau, N., Jézéquel, J.-M.: An eclipse modelling framework alternative to meet the models@runtime requirements. In: Proceedings of the 15th International Conference on Model Driven Engineering Languages and Systems MODELS'12, pp. 87–101. Springer, Berlin (2012)
87. France, R., Rumpe, B.: Model-driven development of complex software: a research roadmap. In: 2007 Future of Software Engineering FOSE '07, pp. 37–54. IEEE Computer Society, Washington, Dc, USA (2007)
88. Frece, A., Juric, M.B.: Modeling functional requirements for configurable content- and context-aware dynamic service selection in business process models. J. Vis. Lang. Comput. **23**(4), 223–247 (2012)
89. Frey, A.G., Calvary, G., Dupuy-Chessa, S.: Users need your models!: exploiting design models for explanations. In: Proceedings of the 26th Annual BCS Interaction Specialist Group Conference on People and Computers, BCS-HCI '12, pp. 79–88. British Computer Society, Swinton (2012)

90. Fuentes, L., Sánchez, P.: Transactions on aspect-oriented software development vi. In: Dynamic Weaving of Aspect-Oriented Executable UML Models, pp. 1–38. Springer, Berlin (2009)
91. Gamez, N., Fuentes, L., Aragüez, M.A.: Autonomic computing driven by feature models and architecture in famiware. In: Proceedings of the 5th European Conference on Software Architecture ECSA'11, pp. 164–179, Springer, Berlin (2011)
92. Gao, Y., Yang, T., Xu, M., Xing, N.: An unsupervised anomaly detection approach for spacecraft based on normal behavior clustering. In: Proceedings of the 2012 Fifth International Conference on Intelligent Computation Technology and Automation ICICTA '12, pp. 478–481. IEEE Computer Society, Washington, DC, USA (2012)
93. García Frey, A.: Self-explanatory user interfaces by model-driven engineering. In: Proceedings of the 2nd ACM SIGCHI Symposium on Engineering Interactive Computing Systems EICS '10, pp. 341–344. ACM, New York (2010)
94. García Frey, A., Céret, E., Dupuy-Chessa, S., Calvary, G., Gabillon, Y.: Usicomp: an extensible model-driven composer. In: Proceedings of the 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems EICS '12, pp . 263–268. ACM, New York (2012)
95. Garlan, D., Cheng, S.-W., Huang, A.-C., Schmerl, B., Steenkiste, P.: Rainbow: architecture-based self-adaptation with reusable infrastructure. Computer **37**(10), 46–54 (2004)
96. Garlan, D., Schmerl, B.: Model-based adaptation for self-healing systems. In: Proceedings of the First Workshop on Self-Healing Systems WOSS '02, pp. 27–32, ACM, New York (2002)
97. Garlan, D., Schmerl, B.: Using architectural models at runtime: research challenges. In: Oquendo, F., Warboys, B., Morrison, R. (eds.) Software Architecture, vol. 3047 of Lecture Notes in Computer Science, pp. 200–205. Springer, Berlin (2004)
98. Garlan, D., Schmerl, B., Chang, J.: Using gauges for architecture-based monitoring and adaptation. In: Proceedings of the Working Conference on Complex and Dynamic Systems Architecture (2001)
99. Garzon, S.R., Cebulla, M.: Model-based personalization within an adaptable human-machine interface environment that is capable of learning from user interactions. In: Proceedings of the 2010 Third International Conference on Advances in Computer-Human Interactions ACHI '10, pp. 191–198. IEEE Computer Society, Washington (2010)
100. Geihs, K., Reichle, R., Khan, M.U., Solberg, A., Hallsteinsen, S.: Model-driven development of self-adaptive applications for mobile devices: (research summary). In: Proceedings of the 2006 International Workshop on Self-Adaptation and Self-Managing Systems SEAMS '06, pp. 95–95. ACM, New York (2006)
101. Georgas, J.C., Hoek, A.V.D., Taylor, R.N.: Using architectural models to manage and visualize runtime adaptation. Computer **42**(10), 52–60 (2009)
102. Ghezzi, C.: The fading boundary between development time and run time. In: Web Services (ECOWS), 2011 Ninth IEEE European Conference on, pp. 11–11 (2011)
103. Ghezzi, C., Mocci, A., Sangiorgio, M.: Runtime monitoring of functional component changes with behavior models. In: Proceedings of the 2011th International Conference on Models in Software Engineering, MODELS'11, pp. 152–166. Springer, Berlin (2012)
104. Ghezzi, C., Tamburrelli, G.: Predicting performance properties for open systems with kami. In: Proceedings of the 5th International Conference on the Quality of Software Architectures: Architectures for Adaptive Software Systems, QoSA '09, pp. 70–85. Springer, Berlin (2009)
105. Giese, H., Hildebrandt, S.: Incremental model synchronization for multiple updates. In: Proceedings of the Third International Workshop on Graph and Model Transformations GRaMoT '08, pp. 1–8. ACM, New York (2008)
106. Giese, H., Lambers, L., Becker, B., Hildebrandt, S., Neumann, S., Vogel, T., Wätzoldt, S.: Graph transformations for mde, adaptation, and models at runtime. In: Proceedings of the 12th International Conference on Formal Methods for the Design of Computer, Communication, and Software Systems: Formal Methods for Model-Driven Engineering SFM'12, pp. 137–191. Springer, Berlin (2012)
107. Giese, H., Wagner, R.: Incremental model synchronization with triple graph grammars. In: Proceedings of the 9th International Conference on Model Driven Engineering Languages and Systems MoDELS'06, pp. 543–557. Springer, Berlin (2006)
108. Giese, H., Wagner, R.: From model transformation to incremental bidirectional model synchronization. Softw. Syst. Model. **8**, 21–43 (2009)
109. Gjerlufsen, T., Ingstrup, M., Wolff, J., Olsen, O.: Mirrors of meaning: supporting inspectable runtime models. Computer **42**(10), 61–68 (2009)
110. Goldsby, H., Cheng, B.H.C., McKinley, P.K., Knoester, D.B., Ofria, C.: Digital evolution of behavioral models for autonomic systems. In: Strassner, J., Dobson, S.A., Fortes, J.A.B., Goswami, K.K. (eds.) ICAC, pp. 87–96. IEEE Computer Society (2008)
111. Goldsby, H., Sawyer, P., Bencomo, N., Cheng, B.H.C., Hughes, D.: Goal-based modeling of dynamically adaptive system requirements. In: Engineering of Computer Based Systems, 2008. ECBS 2008. 15th Annual IEEE International Conference and Workshop on the pp. 36–45 (2008)
112. Goldsby, H.J., Cheng, B.H.: Automatically generating behavioral models of adaptive systems to address uncertainty. In: Proceedings of the 11th International Conference on Model Driven Engineering Languages and Systems MoDELS '08, pp. 568–583, Springer, Berlin (2008)
113. Goldsby, H.J., Cheng, B.H., Zhang, J.: Models in software engineering. In: AMOEBA-RT: Run-Time Verification of Adaptive Software, pp. 212–224. Springer, Berlin (2008)
114. Götz, S., Wilke, C., Cech, S., Aßmann, U.: Architecture and Mechanisms for Energy Auto Tuning. In: Proceeings of Sustainable ICTs and Management Systems for Green Computing (2012)
115. Götz, S., Wilke, C., Schmidt, M., Cech, S., Aßmann, U.: Towards energy auto tuning. In: Proceedings of First Annual International Conference on Green Information Technology (GREEN IT), pp. 122–129 (2010)
116. Grace, P., Blair, G., Samuel, S.: Remmoc: a reflective middleware to support mobile client interoperability, pp. 1170–1187 (2003)
117. Grace, P., Blair, G.S., Cortes, C.F., Bencomo, N.: Engineering complex adaptations in highly heterogeneous distributed systems. In: Proceedings of the 2nd International Conference on Autonomic Computing and Communication Systems Autonomics '08, pp. 27:1–27:10, ICST, Brussels, Belgium, Belgium, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering) (2008)
118. Grace, P., Blair, G.S., Samuel, S.: A reflective framework for discovery and interaction in heterogeneous mobile environments. SIGMOBILE Mob. Comput. Commun. Rev. **9**(1), 2–14 (2005)
119. Grace, P., Bromberg, Y.-D., Réveillère, L., Blair, G.: Overstar: an open approach to end-to-end middleware services in systems of systems. In: Proceedings of the 13th International Middleware Conference, Middleware '12, pp. 229–248. Springer, New York (2012)
120. Grace, P., Truyen, E., Lagaisse, B., Joosen, W.: The case for aspect-oriented reflective middleware. In: Proceedings of the 6th International Workshop on Adaptive and Reflective Middleware: Held at the ACM/IFIP/USENIX International Middleware Conference, p. 2. ACM (2007)

121. Graf, P., Hubner, M., Müller-Glaser, K., Becker, J.: A graphical model-level debugger for heterogenous reconfigurable architectures. In: Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on, pp. 722–725 (2007)
122. Graf, P., Müller-Glaser, K.D.: Dynamic mapping of runtime information models for debugging embedded software. In: Proceedings of the Seventeenth IEEE International Workshop on Rapid System Prototyping RSP '06, pp. 3–9. IEEE Computer Society, Washington, DC, USA (2006)
123. Gu, Z., Wang, S., Kodase, S., Shin, K.G.: Multi-view modeling and analysis of embedded real-time software with meta-modeling and model transformation. In: Proceedings of the Eighth IEEE International Conference on High Assurance Systems Engineering HASE'04, pp. 32–41. IEEE Computer Society, Washington, DC, USA (2004)
124. Haberl, W., Herrmannsdoerfer, M., Birke, J., Baumgarten, U.: Model-level debugging of embedded real-time systems. In: Proceedings of the 2010 10th IEEE International Conference on Computer and Information Technology, CIT '10, pp. 1887–1894. IEEE Computer Society, Washington, DC, USA (2010)
125. Halfond, W.G.J., Orso, A.: Amnesia: analysis and monitoring for neutralizing sql-injection attacks. In: Proceedings of the 20th IEEE/ACM international Conference on Automated Software Engineering ASE '05, pp. 174–183. ACM, New York (2005)
126. Hallsteinsen, S., Stav, E., Solberg, A., Floch, J.: Using product line techniques to build adaptive systems. In: Proceedings of the 10th International on Software Product Line Conference SPLC '06, pp. 141–150. IEEE Computer Society, Washington, DC, USA (2006)
127. Hamann, L., Gogolla, M., Honsel, D.: Towards supporting multiple execution environments for uml/ocl models at runtime. In: Proceedings of the 7th Workshop on Models@run.time, MRT '12, pp. 46–51. ACM, New York (2012)
128. Hamann, L., Gogolla, M., Kuhlmann, M.: Ocl-based runtime monitoring of jvm hosted applications. Electron. Commun. EASST, 44 (2011)
129. Hamann, L., Hofrichter, O., Gogolla, M.: Ocl-based runtime monitoring of applications with protocol state machines. In: Proceedings of the 8th European Conference on Modelling Foundations and Applications ECMFA'12. pp. 384–399, Springer, Berlin (2012)
130. Hao, R., Morin, B., Berre, A.-J.: A semi-automatic behavioral mediation approach based on models@runtime. In: Proceedings of the 7th Workshop on Models@run.time, MRT '12, pp. 67–71, ACM, New York (2012)
131. Hoehndorf, R., Ngomo, A.-C.N., Herre, H.: Developing consistent and modular software models with ontologies. In: Proceedings of the 2009 Conference on New Trends in Software Methodologies, Tools and Techniques: Proceedings of the Eighth SoMeT 09, pp. 399–412. IOS Press, Amsterdam (2009)
132. Holmes, T.: From business application execution to design through model-based reporting. In: Proceedings of the 2012 IEEE 16th International Enterprise Distributed Object Computing Conference EDOC '12, pp. 143–153. IEEE Computer Society, Washington, DC, USA (2012)
133. Holmes, T., Zdun, U., Daniel, F., Dustdar, S.: Monitoring and analyzing service-based internet systems through a model-aware service environment. In: Proceedings of the 22nd International Conference on Advanced Information Systems Engineering, CAiSE'10, pp. 98–112. Springer, Berlin (2010)
134. Holmes, T., Zdun, U., Dustdar, S.: Automating the management and versioning of service models at runtime to support service monitoring. In: Proceedings of the 2012 IEEE 16th International Enterprise Distributed Object Computing Conference, EDOC '12, pp. 211–218. IEEE Computer Society, Washington, DC, USA (2012)
135. Hooman, J., Hendriks, T.: Models in Software Engineering. In: Model-Based Run-Time Error Detection, pp. 225–236. Springer, Berlin (2008)
136. Huang, G., Song, H., Mei, H.: Sm@rt: towards architecture-based runtime management of internetware systems. In: Proceedings of the First Asia-Pacific Symposium on Internetware, Internetware '09, pp. 9:1–9:10, ACM, New York (2009)
137. Huber, N., Brosig, F., Kounev, S.: Modeling dynamic virtualized resource landscapes. In: Proceedings of the 8th International ACM SIGSOFT Conference on Quality of Software Architectures, QoSA '12, pp. 81–90. ACM, New York (2012)
138. Hummer, W., Gaubatz, P., Strembeck, M., Zdun, U., Dustdar, S.: An integrated approach for identity and access management in a soa context. In: Proceedings of the 16th ACM Symposium on Access Control Models and Technologies, SACMAT '11, pp. 21–30, ACM, New York (2011)
139. Ingstrup, M., Hansen, K.M.: A declarative approach to architectural reflection. In: Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture, WICSA '05, pp. 149–158, IEEE Computer Society, Washington (2005)
140. Inverardi, P., Mori, M.: Feature oriented evolutions for context-aware adaptive systems. In: Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE), IWPSE-EVOL '10, pp. 93–97, ACM, New York (2010)
141. Inverardi, P., Mori, M.: Model checking requirements at run-time in adaptive systems. In: Proceedings of the 8th Workshop on Assurances for Self-Adaptive Systems, ASAS '11, pp. 5–9, ACM, New York (2011)
142. Inverardi, P., Mori, M.: Requirements models at run-time to support consistent system evolutions. In: Requirements@Run.Time (RE@RunTime), 2011 2nd International Workshop on, pp. 1–8 (2011)
143. Inverardi, P., Mori, M.: A software lifecycle process to support consistent evolutions. In: Lemos, R., Giese, H., Müller, H., Shaw, M. (eds.) Software Engineering for Self-Adaptive Systems II, vol. 7475 of Lecture Notes in Computer Science, pp. 239–264. Springer, Berlin (2013)
144. Inverardi, P., Pelliccione, P., Tivoli, M.: Towards an assume-guarantee theory for adaptable systems. In: Software Engineering for Adaptive and Self-Managing Systems, 2009. SEAMS '09. ICSE Workshop on, pp. 106–115 (2009)
145. Jiang, M., Zhang, J., Raymer, D., Strassner, J.: A modeling framework for self-healing software systems. In: Workshop "Models@run.time" at the 10th International Conference on model Driven Engineering Languages and Systems (2007)
146. Kappel, G., Wimmer, M., Retschitzegger, W., Schwinger, W.: The evolution of conceptual modeling. In: Leveraging Model-Based Tool Integration by Conceptual Modeling Techniques, pp. 254–284. Springer, Berlin (2011)
147. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. Computer **36**(1), 41–50 (2003)
148. Khan, M., Reichle, R., Geihs, K.: Architectural constraints in the model-driven development of self-adaptive applications. IEEE Distrib. Syst. Online **9**(7), 1–1 (2008)
149. Kindler, E.: Integrating behaviour in software models: an event coordination notation—concepts and prototype. In: Proceedings of the Third Workshop on Behavioural Modelling, BM-FA '11, pp. 41–48. ACM, New York (2011)
150. Kitchenham, B., Charters, S.: Guidelines for Performing Systematic Literature Reviews in Software Engineering. Technical Report EBSE 2007–001, Keele University and Durham University Joint, Report (2007)
151. Kitchenham, B., Pearl Brereton, O., Budgen, D., Turner, M., Bailey, J., Linkman, S.: Systematic literature reviews in software

engineering—a systematic literature review. Inf. Softw. Technol. **51**(1), 7–15 (2009)
152. Kon, F., Costa, F., Blair, G., Campbell, R.H.: The case for reflective middleware. Commun. ACM **45**(6), 33–38 (2002)
153. Kordon, F.: Guest editor's introduction: rapid system prototyping. IEEE Distributed Systems Online **8**(3):7 (2007)
154. Krasnogolowy, A., Hildebrandt, S., Watzoldt, S.: Flexible debugging of behavior models. In: Industrial Technology (ICIT), 2012 IEEE International Conference on pp. 331–336 (2012)
155. Krikava, F., Collet, P.: A reflective model for architecting feedback control systems. In: SEKE, pp. 553–559. Knowledge Systems Institute Graduate School, (2011)
156. Krüger, I.H., Meisinger, M., Menarini, M.: Interaction-based runtime verification for systems of systems integration. J. Log. Comput. **20**(3), 725–742 (2010)
157. Křikava, F., Collet, P., France, R.B.: Actor-based runtime model of adaptable feedback control loops. In: Proceedings of the 7th Workshop on Models@run.time MRT '12, pp. 39–44, ACM, New York (2012)
158. Le Duc, B., Collet, P., Malenfant, J., Rivierre, N.: A qoi-aware framework for adaptive monitoring. In: ADAPTIVE 2010, The Second International Conference on Adaptive and Self-Adaptive Systems and Applications, pp. 133–141 (2010)
159. Lehmann, G., Blumendorf, M., Feuerstack, S., Albayrak, S.: Interactive systems. design, specification, and verification. In: Utilizing Dynamic Executable Models for User Interface Development, pp. 306–309. Springer, Berlin (2008)
160. Lehmann, G., Blumendorf, M., Trollmann, F., Albayrak, S.: Meta-modeling runtime models. In: Proceedings of the 2010 International Conference on Models in Software Engineering, MODELS'10, pp. 209–223, Springer, Berlin (2011)
161. Lehmann, G., Rieger, A., Blumendorf, M., Albayrak, S.: A 3-layer architecture for smart environment models. In: Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on, pp. 636–641 (2010)
162. Litoiu, M., Woodside, M., Zheng, T.: Hierarchical model-based autonomic control of software systems. In: Proceedings of the 2005 Workshop on Design and Evolution of Autonomic Application Software DEAS '05, pp. 1–7, ACM, New York (2005)
163. Liu, C.-S., Li, C.-Y., Tang, J.T., Cheng, Y.: Runtime software architecture model based on design hierarchy. In: Machine Learning and Cybernetics, 2007 International Conference on volume 3, pp. 1444–1449 (2007)
164. Ma, Y., Liu, X., Wu, Y., Grace, P.: Model-based management of service composition. In: Proceedings of the 2013 IEEE Seventh International Symposium on Service-Oriented System Engineering SOSE '13, pp. 103–112. IEEE Computer Society, Washington, DC, USA (2013)
165. MiSE '08: Proceedings of the 2008 International Workshop on Models in Software Engineering. New York, NY, USA (2008). ACM 529080
166. Maes, P.: Concepts and experiments in computational reflection. Sigplan Notices **22**(12), 147–155 (1987)
167. Maoz, S.: Models in software engineering. In: Model-Based Traces, pp. 109–119. Springer, Berlin (2009)
168. Maoz, S.: Using model-based traces as runtime models. Computer **42**(10), 28–36 (2009)
169. Maoz, S., Harel, D.: On tracing reactive systems. Softw. Syst. Model. **10**(4), 447–468 (2011)
170. Matevska, J.: Model-based runtime reconfiguration of component-based systems. In: Proceedings of the Warm Up Workshop for ACM/IEEE ICSE 2010 WUP '09, pp. 33–36. ACM, New York (2009)
171. Mathis, M.M., Kerbyson, D.J.: Dynamic performance prediction of an adaptive mesh application. In: Proceedings of the 20th International Conference on Parallel and Distributed Processing IPDPS'06, pp. 367–367. Washington, DC, USA, IEEE Computer Society (2006)
172. Mayerhofer, T., Langer, P., Kappel, G.: A runtime model for fuml. In: Proceedings of the 7th Workshop on Models@run.time MRT '12, pp. 53–58. ACM, New York (2012)
173. Mayerhofer, T., Langer, P., Wimmer, M.: Towards xmof: executable dsmls based on fuml. In: Proceedings of the 2012 Workshop on Domain-Specific Modeling DSM '12, pp. 1–6. ACM, New York (2012)
174. McQuiggan, S.W., Lester, J.C.: Diagnosing self-efficacy in intelligent tutoring systems: an empirical study. In: Proceedings of the 8th International Conference on Intelligent Tutoring Systems, ITS'06, pp. 565–574. Springer, Berlin (2006)
175. Menascé, D.A., Ruan, H., Gomaa, H.: A framework for qos-aware software components. SIGSOFT Softw. Eng. Notes **29**(1), 186–196 (2004)
176. Miksa, K., Kasztelnik, M., Sabina, P., Walter, T.: Towards semantic modeling of network physical devices. In: Proceedings of the 2009 International Conference on Models in Software Engineering MODELS'09, pp. 329–343, Springer, Berlin (2010)
177. Mocci, A., Sangiorgio, M.: Detecting component changes at run time with behavior models. Computing **95**(3), 191–221 (2013)
178. Morin, B., Barais, O., Jezequel, J.-M., Fleurey, F., Solberg, A.: Models@ run.time to support dynamic adaptation. Computer **42**(10), 44–51 (2009)
179. Morin, B., Barais, O., Marc Jézéquel, J.: K@rt: an aspect-oriented and model-oriented framework for dynamic software product lines. In: Proceedings of the 3rd International Workshop on Models@Runtime, at MoDELS'08 (2008)
180. Morin, B., Barais, O., Nain, G., Jezequel, J.-M.: Taming dynamically adaptive systems using models and aspects. In: Proceedings of the 31st International Conference on Software Engineering ICSE '09, pp. 122–132. IEEE Computer Society, Washington, DC, USA (2009)
181. Morin, B., Fleurey, F., Bencomo, N., Jézéquel, J.-M., Solberg, A., Dehlen, V., Blair, G.: An aspect-oriented and model-driven approach for managing dynamic variability. In: Proceedings of the 11th International Conference on Model Driven Engineering Languages and Systems MoDELS '08, pp. 782–796. Springer, Berlin (2008)
182. Morin, B., Mouelhi, T., Fleurey, F., Le Traon, Y., Barais, O., Jézéquel, J.-M.: Security-driven model-based dynamic adaptation. In: Proceedings of the IEEE/ACM International Conference on Automated Software Engineering, ASE '10, pp. 205–214, ACM, New York (2010)
183. Morris, K.A., Wei, J., Clarke, P.J., Costa, F.M.: Towards adaptable middleware to support service delivery validation in i-dsml execution engines. In: Proceedings of the 2012 IEEE 14th International Symposium on High-Assurance Systems Engineering, HASE '12, pp. 82–89. IEEE Computer Society, Washington, DC, USA (2012)
184. Mos, A., Murphy, J.: Performance management in component-oriented systems using a model driven architecture approach. In: Proceedings of the Sixth International Enterprise Distributed Object Computing Conference (EDOC'02), EDOC '02, p. 227. IEEE Computer Society, Washington, DC, USA (2002)
185. Moser, O., Rosenberg, F., Dustdar, S.: Domain-specific service selection for composite services. IEEE Trans. Softw. Eng. **38**(4), 828–843 (2012)
186. Muller, P.-A., Fleurey, F., Jézéquel, J.-M.: Weaving executability into object-oriented meta-languages. In: Proceedings of the 8th International Conference on Model Driven Engineering Languages and Systems MoDELS'05, pp. 264–278. Springer, Berlin (2005)

187. Murthy, P., Kumar, V.S., Sharma, T., Rao, K.: Quality model driven dynamic analysis. In: Proceedings of the 2011 IEEE 35th Annual Computer Software and Applications Conference COMPSAC '11, pp. 360–365. IEEE Computer Society, Washington, DC, USA (2011)
188. Muskens, J., Chaudron, M.: Integrity management in component based systems. In: Proceedings of the 30th EUROMICRO Conference, EUROMICRO '04, pp. 611–619. IEEE Computer Society, Washington, DC, USA (2004)
189. Nakajima, S.: An architecture of dynamically adaptive php-based web applications. In: Proceedings of the 2011 18th Asia-Pacific Software Engineering Conference, APSEC '11, pp. 203–210. IEEE Computer Society, Washington, DC, USA (2011)
190. Nam, M.-Y., de Niz, D., Wrage, L., Sha, L.: Resource allocation contracts for open analytic runtime models. In: Proceedings of the Ninth ACM International Conference on Embedded Software EMSOFT '11, pp. 13–22. ACM, New York (2011)
191. Nguyen, T., Colman, A.: A feature-oriented approach for web service customization. In: Proceedings of the 2010 IEEE International Conference on Web Services, ICWS '10, pp. 393–400.. IEEE Computer Society, Washington, DC, USA (2010)
192. Nguyen, V.H., Fouquet, F., Plouzeau, N., Barais, O.: A process for continuous validation of self-adapting component based systems. In: Proceedings of the 7th Workshop on Models@run.time MRT '12, pp. 32–37. ACM, New York (2012)
193. Nierstrasz, O., Denker, M., Renggli, L.: Software engineering for self-adaptive systems. In: Model-Centric, Context-Aware Software Adaptation, pp. 128–145. Springer, Berlin (2009)
194. Nilsson, E.G., Floch, J., Hallsteinsen, S., Stav, E.: Using a patterns-based modelling language and a model-based adaptation architecture to facilitate adaptive user interfaces. In: Proceedings of the 13th International Conference on Interactive Systems: Design, Specification, and Verification, DSVIS'06, pp. 234–247. Springer, Berlin (2007)
195. Nordstrom, S., Dubey, A., Keskinpala, T., Datta, R., Neema, S., Bapty, T.: Model predictive analysis for autonomicworkflow management in large-scale scientific computing environments. In: Proceedings of the Fourth IEEE International Workshop on Engineering of Autonomic and Autonomous Systems EASE '07, pp. 37–42. IEEE Computer Society, Washington, DC, USA (2007)
196. Occello, A., Pinna-Déry, A.-M., Riveill, M.: A runtime model for monitoring software adaptation safety and its concretisation as a service. In: Bencomo, N., Blair, G., France, R., Freddy, M., Cedric, J. (eds.) Models@runtime(MRT08), pp. 67–76, Toulouse, France (2008)
197. Olivier-Nathanaël, B.D., Benoit, B.: Toward a model-driven access-control enforcement mechanism for pervasive systems. In: Proceedings of the Workshop on Model-Driven Security, MDsec '12, pp. 6:1–6:6. ACM, New York (2012)
198. Oreizy, P., Gorlick, M.M., Taylor, R.N., Heimbigner, D., Johnson, G., Medvidovic, N., Quilici, A., Rosenblum, D.S., Wolf, A.L.: An architecture-based approach to self-adaptive software. IEEE Intell. Syst. **14**(3), 54–62 (1999)
199. Oreizy, P., Medvidovic, N., Taylor, R.N.: Architecture-based runtime software evolution. In: Proceedings of the 20th International Conference on Software Engineering ICSE '98, pp. 177–186. IEEE Computer Society, Washington, DC, USA (1998)
200. Ortiz, O., García, A.B., Capilla, R., Bosch, J., Hinchey, M.: Runtime variability for dynamic reconfiguration in wireless sensor network product lines. In: Proceedings of the 16th International Software Product Line Conference, vol. 2, SPLC '12, pp. 143–150. ACM, New York (2012)
201. Parra, C., Blanc, X., Duchien, L.: Context awareness for dynamic service-oriented product lines. In: Proceedings of the 13th International Software Product Line Conference, SPLC '09, pp. 131–140. Carnegie Mellon University, Pittsburgh, PA, USA (2009)
202. Perrouin, G., Morin, B., Chauvel, F., Fleurey, F., Klein, J., Le Traon, Y., Barais, O., Jézéquel, J.-M.: Towards flexible evolution of dynamically adaptive systems. In: Proceedings of the 2012 International Conference on Software Engineering ICSE 2012, pp. 1353–1356. IEEE Press, Piscataway, NJ, USA (2012)
203. Pienaar, J.A., Raghunathan, A., Chakradhar, S.: Mdr: performance model driven runtime for heterogeneous parallel platforms. In: Proceedings of the International Conference on Supercomputing ICS '11, pp. 225–234. ACM, New York (2011)
204. Pleumann, J., Haustein, S.: A model-driven runtime environment for web applications. In: Proceedings of UML 2003—The Unified Modeling Language. Model Languages and Applications. 6th International Conference, pp. 190–204. Springer, Berlin (2003)
205. Porcarelli, S., Castaldi, M., Giandomenico, F.D., Bondavalli, A., Inverardi, P.: A framework for reconfiguration-based fault-tolerance in distributed systems. In: Distributed Systems, Architecting Dependable Systems II. Springer, Berlin (2004)
206. Rammig, F.J., Zhao, Y., Samara, S.: On-line model checking as operating system service. In: Proceedings of the 7th IFIP WG 10.2 International Workshop on Software Technologies for Embedded and Ubiquitous Systems, SEUS '09, pp. 131–143. Springer, Berlin (2009)
207. Redlich, D., Gilani, W.: Event-driven process-centric performance prediction via simulation. In: Business Process Management Workshops, pp. 473–478. Springer, Berlin (2012)
208. Robinson, H., Segal, J., Sharp, H.: Ethnographically-informed empirical studies of software practice. Inf. Softw. Technol. **49**(6), 540–551, Qualitative Software Engineering Research (2007)
209. Rodríguez-Gracia, D., Criado, J., Iribarne, L., Padilla, N., Vicente-Chicote, C.: Runtime adaptation of architectural models: an approach for adapting user interfaces. In: Proceedings of the 2nd International Conference on Model and Data Engineering, MEDI'12, pp. 16–30. Springer, Berlin (2012)
210. Rohr, M., Boskovic, M., Giesecke, S., Hasselbring, W.: Model-driven development of selfmanaging software systems. In: ACM/IEEE MoDELS Workshop on Models@Runtime
211. Rosenmüller, M., Siegmund, N., Apel, S., Saake, G.: Flexible feature binding in software product lines. Autom. Softw. Eng. **18**(2), 163–197 (2011)
212. Röttger, S., Zschaler, S.: Tool support for refinement of non-functional specifications. Softw. Syst. Model. **6**(2), 185–204 (2007)
213. SEAMS '11: Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. ACM, New York (2011)
214. Sawyer, P., Bencomo, N., Hughes, D., Grace, P., Goldsby, H.J., Cheng, B.H.C.: Visualizing the analysis of dynamically adaptive systems using i* and dsls. In: Proceedings of the Second International Workshop on Requirements Engineering Visualization REV '07, p. 3, IEEE Computer Society, Washington, DC, USA (2007)
215. Sawyer, P., Bencomo, N., Whittle, J., Letier, E., Finkelstein, A.: Requirements-aware systems: a research agenda for re for self-adaptive systems. In: Requirements Engineering Conference (RE), 2010 18th IEEE International pp. 95–103 (2010)
216. Schlegel, T., Pietschmann, S.: Model-based interactive ubiquitous systems. In: Proceedings of the 3rd ACM SIGCHI Symposium on Engineering Interactive Computing Systems, EICS '11, pp. 337–338. ACM, New York (2011)
217. Schmerl, B., Aldrich, J., Garlan, D., Kazman, R., Yan, H.: Discovering architectures from running systems. IEEE Trans. Softw. Eng. **32**(7), 454–466 (2006)
218. Schmidt, D.C.: Guest editor's introduction: model-driven engineering. Computer **39**(2), 25–31 (2006)
219. Schneider, D., Trapp, M.: Runtime safety models in open systems of systems. In: Proceedings of the 2009 Eighth IEEE International

Conference on Dependable, Autonomic and Secure Computing DASC '09, pp. 455–460. IEEE Computer Society, Washington, DC, USA (2009)
220. Schneider, D., Trapp, M.: Conditional safety certification of open adaptive systems. ACM Trans. Auton. Adapt. Syst. **8**(2), 8:1–8:20 (2013)
221. Schonbock, J., Kappel, G., Wimmer, M., Kusel, A., Retschitzegger, W., Schwinger, W.: Debugging model-to-model transformations. In: Proceedings of the 2012 19th Asia-Pacific Software Engineering Conference—Volume 01 APSEC '12, pp. 164–173. IEEE Computer Society, Washington, DC, USA (2012)
222. Schwartze, V., Blumendorf, M., Albayrak, S.: Adjustable context adaptations for user interfaces at runtime. In: Proceedings of the International Conference on Advanced Visual Interfaces, AVI '10, pp. 321–324. ACM, New York (2010)
223. Serral, E., Valderas, P., Pelechano, V.: Automating routine tasks in ami systems by using models at runtime. In: Proceedings of the First International Joint Conference on Ambient Intelligence Am I'10, pp. 1–10, Springer, Berlin (2010)
224. Serral, E., Valderas, P., Pelechano, V.: Supporting runtime system evolution to adapt to user behaviour. In: Proceedings of the 22nd International Conference on Advanced Information Systems Engineering, CAiSE'10, pp. 378–392. Springer, Berlin (2010)
225. Serral, E., Valderas, P., Pelechano, V.: Context-adaptive coordination of pervasive services by interpreting models during runtime. Comput. J. **56**(1), 87–114 (2013)
226. Shroff, G., Agarwal, P., Devanbu, P.: Instant multi-tier web applications without tears. In: Proceedings of the 2nd India Software Engineering Conference ISEC '09, pp. 3–12, ACM, New York (2009)
227. Silva Souza, V.E., Lapouchnian, A., Robinson, W.N., Mylopoulos, J.: Awareness requirements for adaptive systems. In: Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS '11, pp. 60–69. ACM, New York (2011)
228. Simmonds, J., Ben-David, S., Chechik, M.: The smart internet. In: Monitoring and Recovery of Web Service Applications, pp. 250–288. Springer, Berlin (2010)
229. Sindhgatta, R., Sengupta, B.: An extensible framework for tracing model evolution in soa solution design. In: Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications OOPSLA '09, pp. 647–658. ACM, New York (2009)
230. Song, H., Gallagher, M., Clarke, S.: Rapid gui development on legacy systems: a runtime model-based solution. In: Proceedings of the 7th Workshop on Models@run.time MRT '12, pp. 25–30. ACM, New York (2012)
231. Song, H., Huang, G., Chauvel, F., Sun, Y., Mei, H.: Sm@rt: representing run-time system data as mof-compliant models. In: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering—Volume 2 ICSE '10, pp. 303–304. ACM, New York (2010)
232. Song, H., Huang, G., Chauvel, F., Xiong, Y., Hu, Z., Sun, Y., Mei, H.: Supporting runtime software architecture: a bidirectional-transformation-based approach. J. Syst. Softw. **84**(5), 711–723 (2011)
233. Song, H., Huang, G., Chauvel, F., Zhang, W., Sun, Y., Shao, W., Mei, H.: Instant and incremental qvt transformation for runtime models. In: Proceedings of the 14th International Conference on Model Driven Engineering Languages and Systems, MODELS'11, pp. 273–288. Springer, Berlin (2011)
234. Song, H., Huang, G., Xiong, Y., Chauvel, F., Sun, Y., Mei, H.: Inferring meta-models for runtime system data from the clients of management apis. In: Proceedings of the 13th International Conference on Model Driven Engineering Languages and Systems: Part II MODELS'10, pp. 168–182, Springer, Berlin (2010)
235. Song, H., Xiong, Y., Chauvel, F., Huang, G., Hu, Z., Mei, H.: Generating synchronization engines between running systems and their model-based views. In: Proceedings of the 2009 International Conference on Models in Software Engineering MODELS'09, pp. 140–154, Springer, Berlin (2010)
236. Sottet, J.-S.: Ingénierie dirigée par les modèles pour la plasticité des interfaces homme machine. In: Proceedings of the 19th International Conference of the Association Francophone d'Interaction Homme-Machine IHM '07, pp. 253–256, ACM, New York (2007)
237. Sottet, J.-S., Ganneau, V., Calvary, G., Coutaz, J., Demeure, A., Favre, J.-M., Demumieux, R.: Model-driven adaptation for plastic user interfaces. In: Proceedings of the 11th IFIP TC 13 International Conference on Human-Computer Interaction INTERACT'07, pp. 397–410. Springer, Berlin (2007)
238. Sousa, G.C.M., Costa, F.M., Clarke, P.J., Allen, A.A.: Model-driven development of dsml execution engines. In: Proceedings of the 7th Workshop on Models@run.time MRT '12, pp. 10–15. ACM, New York (2012)
239. Spieker, M., Noyer, A., Iyenghar, P., Bikker, G., Wuebbelmann, J., Westerkamp, C.: Model based debugging and testing of embedded systems without affecting the runtime behaviour. In: Emerging Technologies Factory Automation (ETFA), 2012 IEEE 17th Conference on, pp. 1–6 (2012)
240. Steck, A., Lotz, A., Schlegel, C.: Model-driven engineering and run-time model-usage in service robotics. In: Proceedings of the 10th ACM International Conference on Generative Programming and Component Engineering GPCE '11, pp. 73–82, ACM, New York (2011)
241. Steck, A., Schlegel, C.: Managing execution variants in task coordination by exploiting design-time models at run-time. In: Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on, pp. 2064–2069 (2011)
242. Stein, S., Neukirchner, M., Schrom, H., Ernst, R.: Consistency challenges in self-organizing distributed hard real-time systems. In: Proceedings of the 2010 13th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops, ISORCW '10, pp. 39–48. IEEE Computer Society, Washington, DC, USA (2010)
243. Steinbauer, G., Mörth, M., Wotawa, F.: Robocup 2005. In: Real-Time Diagnosis and Repair of Faults of Robot Control Software, pp. 13–23. Springer, Berlin (2006)
244. Strembeck, M., Mendling, J.: Modeling process-related rbac models with extended uml activity models. Inf. Softw. Technol. **53**(5), 456–483 (2011)
245. Sukaviriya, P., Foley, J.D., Griffith, T.: A second generation user interface design environment: the model and the runtime architecture. In: Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems, CHI '93, pp. 375–382. ACM, New York (1993)
246. Taconet, C., Kazi-Aoul, Z., Zaier, M., Conan, D.: Ca3m: a runtime model and a middleware for dynamic context management. In: Proceedings of the Confederated International Conferences, CoopIS, DOA, IS, and ODBASE 2009 on On the Move to Meaningful Internet Systems: Part I, OTM '09, pp. 513–530. Springer, Berlin (2009)
247. Tan, L., Kim, J., Sokolsky, O., Lee, I.: Model-based testing and monitoring for hybrid embedded systems. In: Zhang, D., Gregoire, E., DeGroot, D. (eds.) IRI, pp. 487–492. IEEE Systems, Man, and Cybernetics Society (2004)
248. Thonhauser, M., Kreiner, C., Leitner, A.: A model-based architecture supporting virtual organizations in pervasive systems. In: Proceedings of the 2010 15th IEEE International Conference on Engineering of Complex Computer Systems ICECCS '10, pp. 249–252. IEEE Computer Society, Washington, DC, USA (2010)

249. Thonhauser, M., Kreiner, C., Schmid, M.: Interpreting model-based components for information systems. In: Proceedings of the 2009 16th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems, ECBS '09, pp. 254–261. IEEE Computer Society, Washington, DC, USA (2009)
250. Thonhauser, M., Krenn, U., Kreiner, C.: Applying multi-model based components for virtual organizations. In: Proceedings of the 2011 19th International Euromicro Conference on Parallel, Distributed and Network-Based Processing PDP '11, pp. 285–292. IEEE Computer Society, Washington, DC, USA (2011)
251. Tombelle, C., Vanwormhoudt, G.: Dynamic and generic manipulation of models: from introspection to scripting. In: Proceedings of the 9th International Conference on Model Driven Engineering Languages and Systems MoDELS'06, pp. 395–409, Springer, Berlin (2006)
252. Truyen, E., Janssens, N., Sanen, F., Joosen, W.: Support for distributed adaptations in aspect-oriented middleware. In: Proceedings of the 7th International Conference on Aspect-Oriented Software Development, AOSD '08, pp. 120–131. ACM, New York (2008)
253. Uttamchandani, S.: Polus: a self-evolving model-based approach for automating the observe-analyze-act loop. PhD thesis, Champaign, IL, USA, (2005) AAI3199160
254. Vanderdonckt, J., Guerrero-Garcia, J., Gonzalez-Calleros, J.M.: A model-based approach for developing vectorial user interfaces. In: Proceedings of the 2009 Latin American Web Congress (la-web 2009), LA-WEB '09, pp. 52–59. IEEE Computer Society, Washington, DC, USA (2009)
255. Vogel, T., Giese, H.: Adaptation and abstract runtime models. In: Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems SEAMS '10, pp. 39–48. ACM, New York (2010)
256. Vogel, T., Giese, H.: A language for feedback loops in self-adaptive systems: executable runtime megamodels. In: Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2012 ICSE Workshop on, pp. 129–138 (2012)
257. Vogel, T., Neumann, S., Hildebrandt, S., Giese, H., Becker, B.: Model-driven architectural monitoring and adaptation for autonomic systems. In: Proceedings of the 6th International Conference on Autonomic Computing, ICAC '09, pp. 67–68. ACM, New York (2009)
258. Vogel, T., Neumann, S., Hildebrandt, S., Giese, H., Becker, B.: Incremental model synchronization for efficient run-time monitoring. In: Proceedings of the 2009 International Conference on Models in Software Engineering, MODELS'09, pp. 124–139. Springer, Berlin (2010)
259. Vogel, T., Seibel, A., Giese, H.: The role of models and megamodels at runtime. In: Proceedings of the 2010 International Conference on Models in Software Engineering, MODELS'10, pp. 224–238. Springer, Berlin (2011)
260. Vrbaski, M., Mussbacher, G., Petriu, D., Amyot, D.: Goal models as run-time entities in context-aware systems. In: Proceedings of the 7th Workshop on Models@run.time MRT '12, pp. 3–8, ACM, New York (2012)
261. Waignier, G., Meur, A.-F., Duchien, L.: A model-based framework to design and debug safe component-based autonomic systems. In: Proceedings of the 5th International Conference on the Quality of Software Architectures: Architectures for Adaptive Software Systems, QoSA '09. Springer, Berlin (2009)
262. Wang, B., Zhou, X., Yang, G., Yang, Y.: Web services trustworthiness evaluation based on fuzzy cognitive maps. In: Proceedings of the 2010 International Symposium on Intelligence Information Processing and Trusted Computing, IPTC '10, pp. 230–233. IEEE Computer Society, Washington, DC, USA (2010)
263. Wang, L., Wong, E., Xu, D.: A threat model driven approach for security testing. In: Proceedings of the Third International Workshop on Software Engineering for Secure Systems SESS '07, p. 10. IEEE Computer Society, Washington, DC, USA (2007)
264. Weiss, G., Becker, K., Kamphausen, B., Radermacher, A., Gerard, S.: Model-driven development of self-describing components for self-adaptive distributed embedded systems. In: Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on, pp. 477–484 (2011)
265. Welsh, K., Sawyer, P.: Managing testing complexity in dynamically adaptive systems: a model-driven approach. In: Proceedings of the 2010 Third International Conference on Software Testing, Verification, and Validation Workshops ICSTW '10, pp. 290–298. IEEE Computer Society, Washington, DC, USA (2010)
266. Welsh, K., Sawyer, P., Bencomo, N.: Towards requirements aware systems: Run-time resolution of design-time assumptions. In: Automated Software Engineering (ASE), 2011 26th IEEE/ACM International Conference on, pp. 560–563 (2011)
267. Whittle, J., Sawyer, P., Bencomo, N., Cheng, B.H.C., Bruel, J.-M.: Relax: incorporating uncertainty into the specification of self-adaptive systems. In: Proceedings of the 2009 17th IEEE International Requirements Engineering Conference, RE, RE '09, pp. 79–88. IEEE Computer Society, Washington, DC, USA (2009)
268. Wimmer, M., Kappel, G., Schoenboeck, J., Kusel, A., Retschitzegger, W., Schwinger, W.: A petri net based debugging environment for qvt relations. In: Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering ASE '09, pp. 3–14. IEEE Computer Society, Washington, DC, USA (2009)
269. Wimmer, M., Kusel, A., Schoenboeck, J., Kappel, G., Retschitzegger, W., Schwinger, W.: Reviving qvt relations: model-based debugging using colored petri nets. In: Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems MODELS '09, pp. 727–732. Springer, Berlin (2009)
270. Witt, H., Nicolai, T., Kenn, H.: The wui-toolkit: a model-driven ui development framework for wearable user interfaces. In: Proceedings of the 27th International Conference on Distributed Computing Systems Workshops, ICDCSW '07, p. 43, IEEE Computer Society, Washington, DC, USA (2007)
271. Wolfe, C., Graham, T.C., Phillips, W.G.: An incremental algorithm for high-performance runtime model consistency. In: Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems MODELS '09, pp. 357–371. Springer, Berlin (2009)
272. Xiao, L.: An adaptive security model using agent-oriented mda. Inf. Softw. Technol. **51**(5), 933–955 (2009)
273. Yang, R., Chen, Z., Xu, B., Wong, W.E., Zhang, J.: Improve the effectiveness of test case generation on efsm via automatic path feasibility analysis. In: Proceedings of the 2011 IEEE 13th International Symposium on High-Assurance Systems Engineering HASE '11, pp. 17–24. IEEE Computer Society, Washington, DC, USA (2011)
274. Yu, E., Mylopoulos, J.: Why goal-oriented requirements engineering. In: Proceedings of the 4th International Workshop on Requirements Engineering: Foundations of Software, Quality pp. 15–22 (1998)
275. Zeng, K., Guo, Y., Angelov, C.K.: Graphical model debugger framework for embedded systems. In: Proceedings of the Conference on Design, Automation and Test in Europe, DATE '10, pp. 87–92, 3001 Leuven, Belgium, Belgium, European Design and Automation Association (2010)
276. Zeng, L., Lei, H., Dikun, M., Chang, H., Bhaskaran, K.: Model-driven business performance management. In: Proceedings of the IEEE International Conference on e-Business Engineering, ICEBE '05, pp. 295–304. IEEE Computer Society, Washington, DC, USA (2005)

277. Zhang, J., Cheng, B.H.C.: Model-based development of dynamically adaptive software. In: Proceedings of the 28th International Conference on Software Engineering ICSE '06, pp. 371–380, ACM, New York (2006)
278. Zhang, L., Sun, Y., Song, H., Wang, W., Huang, G.: Detecting anti-patterns in java ee runtime system model. In: Proceedings of the Fourth Asia-Pacific Symposium on Internetware, Internetware '12, pp. 21:1–21:8, ACM, New York (2012)
279. Zhao, Y., Oberthür, S., Kardos, M., Rammig, F.J.: Model-based runtime verification framework for self-optimizing systems. Electron. Notes Theor. Comput. Sci. **144**(4), 125–145 (2006)
280. Zhao, Y., Oberthür, S., Montealegre, N., Rammig, F.J., Kardos, M.: Increasing dependability by means of model-based acceptance test inside rtos. In: Proceedings of the 6th International Conference on Parallel Processing and Applied Mathematics PPAM'05, pp. 1034–1041, Springer, Berlin (2006)
281. Zhao, Y., Rammig, F.: Model-based runtime verification framework. Electron. Notes Theor. Comput. Sci. **253**(1), 179–193 (2009)
282. Zimmermann, O.: Architectural decisions as reusable design assets. IEEE Softw. **28**(1), 64–69 (2011)
283. Zinky, J.A., Loyall, J.P., Shapiro, R.: Runtime performance modeling and measurement of adaptive distributed object applications. In: On the Move to Meaningful Internet Systems, 2002—DOA/CoopIS/ODBASE 2002 Confederated International Conferences DOA, CoopIS and ODBASE 2002, pp. 755–772. Springer, London (2002)

Author Biographies

Michael Szvetits is a research associate in the department of software engineering at the University of Applied Sciences Wiener Neustadt who started his academic career as tutor for mathematics and programming. He received his bachelor's degree in 2010 and his master's degree in 2012. He is currently working on his doctorate at the University of Vienna where his research focuses on software modelling, domain-specific languages, model-driven development, and model-based analysis of running systems.

Uwe Zdun is a full professor for software architecture at the Faculty of Computer Science, University of Vienna. He received his doctoral degree from the University of Essen in 2002. His research focuses on modelling of complex software systems, service-oriented systems, architectural decision, software patterns, domain-specific languages, and model-driven development. Uwe has published more than 140 articles in peer-reviewed journals, conferences, book chapters, and workshops and is co-author of the books "Remoting Patterns - Foundations of Enterprise, Internet, and Realtime Distributed Object Middleware", "Process-Driven SOA - Proven Patterns for Business-IT Alignment", and "Software-Architektur." He has participated in 16 R&D projects. Uwe is the editor of the journal Transactions on Pattern Languages of Programming (TPLoP) published by Springer, and the Associate Editor-in-Chief for design and architecture for the IEEE Software magazine.