A Feature-based Survey of Model View Approaches

Hugo Bruneliere, Erik Burger, Jordi Cabot, Manuel Wimmer

Software and Systems Modeling (SoSyM) manuscript No.
(will be inserted by the editor)

A Feature-based Survey of Model View Approaches

Hugo Bruneliere · Erik Burger · Jordi Cabot · Manuel Wimmer

Received: date / Accepted: date

Abstract When dealing with complex systems, information is very often fragmented across many different models expressed within a variety of (modeling) languages. To provide the relevant information in an appropriate way to different kinds of stakeholders, (parts of) such models have to be combined and potentially revamped by focusing on concerns of particular interest for them.

This work has been partially funded by the MoNoGe national collaborative project (French FUI #15), the Electronic Component Systems for European Leadership (ECSEL) Joint Undertaking & the European Union's Horizon 2020 research/innovation program under grant agreement No. 737494 (MegaM@Rt2 project), the Spanish government (TIN2016-75944-R project), the Austrian Federal Ministry of Science, Research and Economy (BMWFW) and National Foundation for Research, Technology and Development. We also would like to thank the various academics and industrials that provided valuable feedback to us on our descriptions of their respective solutions.

Hugo Bruneliere
AtlanModels Team (IMT Atlantique, Inria & LS2N)
IMT Atlantique Bretagne-Pays de la Loire
Nantes, France
E-mail: hugo.bruneliere@imt-atlantique.fr

Erik Burger
Institute for Program Structures and Data Organization
Karlsruhe Institute of Technology (KIT)
Karlsruhe, Germany
E-mail: burger@kit.edu

Jordi Cabot
ICREA
Open University of Catalonia
Barcelona, Spain
E-mail: jordi.cabot@icrea.cat

Manuel Wimmer
CDL-MINT
TU Wien
Vienna, Austria
E-mail: wimmer@big.tuwien.ac.at

Thus, mechanisms to define and compute views over models are highly needed. Several approaches have already been proposed to provide (semi-)automated support for dealing with such *model views*. This paper provides a detailed overview of the current state-of-the-art in this area. To achieve this, we relied on our own experiences of designing and applying such solutions in order to conduct a literature review on this topic. As a result, we discuss the main capabilities of existing approaches and propose a corresponding research agenda. We notably contribute a feature model describing what we believe to be the most important characteristics of the support for views on models. We expect this work to be helpful to both current and potential future users and developers of model view techniques, as well as to any person generally interested in model-based software and systems engineering.

Keywords Modeling · Viewpoint · View · Model · Survey

1 Introduction

When developing modern software systems, models that conform to different modeling languages (i.e., metamodels) are used by engineers and architects to describe the system under development from various perspectives. This often leads to scattering of information across (possibly many) heterogeneous models, and to overlappings/redundancies that can create inconsistencies in the system description [29]. Thus, engineers may have difficulties to understand efficiently the complete system description by looking at all these models in full detail.

This is even a major concern when dealing with systems of systems or cyber-physical systems [24, 67], e.g., especially in an industrial context where multi-disciplinary engineering takes place [28]. In such cases, models actually need to cover hardware infrastructure, deployment, usage scenarios as well as non-functional properties of the system (e.g., performance, reliability, and maintainability). They also need to deal with the involved physical entities such as energy grids, networks, production plants and/or various kinds of IoT devices. All these models have strong dependencies and interconnections to the software models we are familiar with. They are generally combined altogether in order to better monitor and analyze these systems at runtime.

View-based approaches in software engineering have been proposed to tackle these issues [29]. They mostly follow a strategy of proposing a fixed set of predefined *viewpoints* to be used in different application domains or scenarios. This happens for instance with most architectural frameworks such as Zachman [70] or RM-ODP [47], each viewpoint targeting particular perspectives of the system to be considered. It offers several advantages such as improved understanding and more integrated and comprehensive tool support. However, these approaches lack the flexibility required in many scenarios where the useful model views go beyond a limited set of viewpoints, and may change over time as the development process moves on.

Recent advances in model-based engineering have fostered the possibility of having more flexible view-based approaches, based on metamodeling and model transformation techniques notably. The notion of ad-hoc views computed via queries has been studied intensively in past decades. While the metaphor helps developers and modelers to understand these concepts, similar problems such as (incremental) view updates arise. These *model view* approaches usually allow creating custom (semi-)automatically generated views over possibly heterogeneous models. Such a capability can notably help to reduce accidental complexity in software and systems development, deployment and usage.

There are many real-world use cases in which the direct support for such *model views* is actually required. In addition, there are also a lot of scenarios in which the use of model views can be beneficial as part of a larger model-based solution to a given complex problem. From our own practical experiences within various collaborative industrial projects in the past years, we have already observed relevant applications in areas such as (meta-)model federation/integration (e.g., in an Enterprise Architecture (EA) context, cf. also Subsection 2.2), reverse engineering [64], language maintenance and evolution [14] or security [50], to mention just a few. However, each scenario requires a different trade-off in terms of model view capabilities. For instance, a good expressivity of the view definition and an efficient data synchronization support are fundamental to EA model federation or language evolution cases. In the context of reverse engineering or security, scalability in the computation appears to be more important due to the potential handling of very large models. Given the plethora of existing approaches—most of them only providing partial solutions, it is difficult to know how each approach compares to the others. From such scattered information, it is also complicated to identify which one(s) may be better suited for given needs. Thus, our intent with this paper is to provide orientation by proposing a detailed study of the state-of-the-art in this domain. Hoping to stimulate further discussions, we contribute to the modeling community a summary of what we believe to be the main characteristics of model view approaches and their current/future challenges.

The rest of the paper is organized as follows. We have first synthesized a common terminology and selected an illustrative example which we present in Section 2. In Section 3, we describe the method we have used for searching, identifying and selecting relevant approaches. We contribute in Section 4 a feature model summarizing the main characteristics of approaches for views on models. Then, we further use the proposed terminology and feature model to describe and compare the selected approaches in Section 5. Based on this, we identify open research areas and interesting challenges related to the development or use of model views in Section 6 . The paper closes with further discussion on the related work in Section 7 and a general conclusion in Section 8.

2 Terminology and Illustrative Example

In the modeling domain, the terms *view*, *viewpoint* or *viewtype* are used in several different ways. From the very early viewpoint approaches [29, 69] up to the ISO standard 42010 [1], various definitions for these terms have been given. In this present paper, we have decided to follow the definitions of Goldschmidt et al. [30].

In our understanding, a *view* is usually a special kind of model. A view contains information that is related to and coming from other models, which can also be themselves other views. A view is always a view on something. Thus, in an engineering context, the set of physical and/or logical entities that a view represents is called a *system*. Such a system can be observed from different *viewpoints*, each of them providing different perspectives over it. The relation between views and other models is specified by various means such as transformations, rules, queries, or other formalisms. As any model, a view conforms to a metamodel, which is usually called *viewtype*. This viewtype can be defined a-priori, or can be sometimes deduced from the specification of the view itself. This is however not the case in general, as viewtype and view specifications are usually clearly separated. Such a situation can be practically observed in many of the approaches that we will evaluate later on in this paper. The goal of this section is to clarify all these concepts and illustrate them, before we go deeper into the selection and study of model views approaches.

2.1 General Definitions

We propose some general definitions for the most frequently encountered terms while searching for and studying solutions for views/viewpoints on models. These main terms are graphically summarized in Figure 1 and textually explained in the following:

- A **system** is a unit consisting of multiple interdependent components, which are designed and implemented by engineers. A system encompasses software, hardware, requirements, as well as all other artefacts created during its development process.
- A **viewpoint** is the description of a combination, partitioning and/or restriction of concerns from which systems can be observed. In our modeling context, it consists of a set of concepts coming from one or more metamodels, eventually complemented with some new interconnections between them and newly added features (cf. **viewtype** definition).
- A **viewtype** is a metamodel that describes the types of elements that can appear in a view, i.e., the formalism/language actually used. An element in a viewtype may be part of one of the base metamodels, or may be specifically defined for the viewtype. A given viewtype can be relevant for several viewpoints, and a viewpoint usually defines several viewtypes.

- A **view** is a representation of a specific system from the perspective of a given viewpoint. In our modeling context, it is an instance of a particular viewtype and consists of a set of elements coming from one or more base models. It is eventually complemented with some new interconnections between them and additional data, that are manually entered and/or computed automatically (usually via one or more model transformations).
- A **base metamodel** is a metamodel that is used in and referenced from a given viewtype definition. Depending on the approaches, a viewtype specification can possibly have one or several different base metamodels.
- A **base model** is a model that is used in and referenced from a given view. Depending on approaches and on the corresponding defined viewpoint (and related viewtypes), a view can possibly gather elements coming from one or more base models.

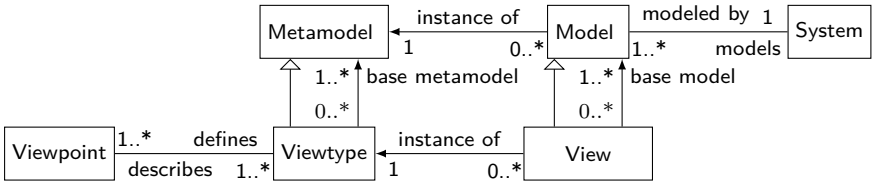

Fig. 1 View Terminology

2.2 An Illustrative Example: Modeling Enterprise Architectures

To concretely illustrate the previously presented terminology, and also to further motivate on the actual need for model views, in this subsection we describe an industrial case for model views.

The TOGAF Enterprise Architecture Platform (TEAP) project[1] was a joint collaboration between the companies Capgemini (IT consulting), DCNS (Naval defense and energy), Obeo (software company specialized in MDE) and the AtlanMod research team. The main objective of the project was to provide an enhanced support for the improved governance of enterprise architectures (EAs).

Obeo is an EA solution provider with its SmartEA tool[2] supporting TOGAF[3]. In the context of the project, DCNS identified the need for specializing the SmartEA standard toolkit according to the particular type of systems of systems they are dealing with. They wanted to extend TOGAF to include both business process information defined using the BPMN[4] standard and re-

[1] http://www.teap-project.org
[2] http://www.obeosmartea.com
[3] https://www.opengroup.org/togaf
[4] http://www.bpmn.org

quirement specifications defined with the ReqIf[5] standard. Besides the need for integrating BPMN and ReqIf models as part of their overall EA solution, DCNS wanted to be able to interconnect these models with the TOGAF models to provide partial views on these combined base models depending on some DCNS member profiles (e.g., for security constraints, access to sensible parts of the base models needed to be filtered out in the views for people without proper privileges). In order to do so, they needed to specify several viewtypes notably linking these TOGAF, ReqIF, and BPMN base metamodels altogether. Figure 2 shows a partial realization of that.

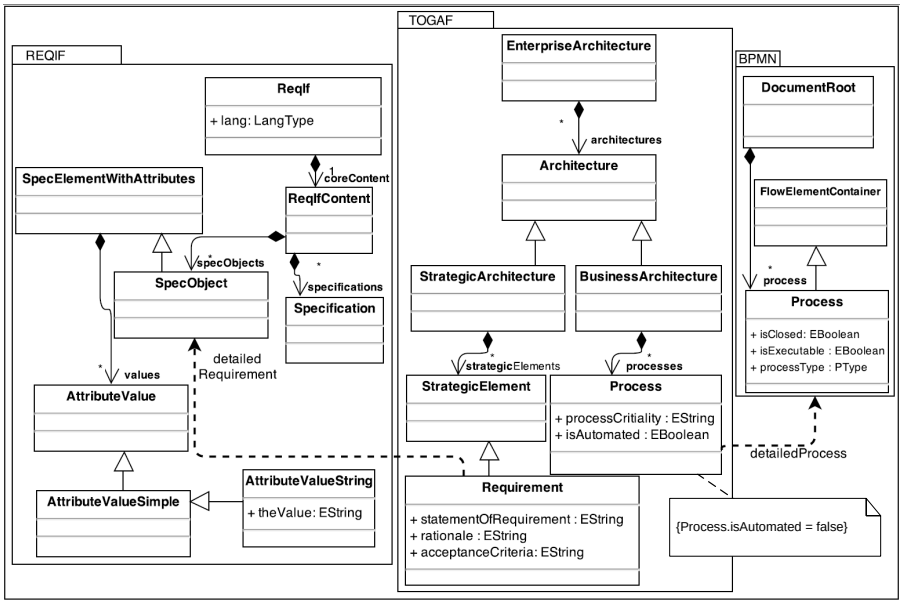

Fig. 2 Example of a viewtype combining the TOGAF base metamodel with the ReqIF and BPMN ones (excerpt).

In the TOGAF implementation within SmartEA, business processes and requirements are modeled at a high-level of abstraction as part of the business and strategic architectures respectively. In the ReqIF standard, requirements (called *SpecObject*) are modeled in more detail. The BPMN standard also allows describing detailed business processes (called *Process*). Thanks to the specified view type, these two types of model elements can be directly connected to their TOGAF equivalents (*Requirement* and *Process* respectively) by using the new *detailedRequirement* and *detailedProcess* relations. By following the general idea of *select-project-join* queries: (*i*) via *select*, we are only getting *Processes* which have the *isAutomated* attribute set to *false*, as shown in the corresponding note; (*ii*) via *project*, we are only keeping some

[5] http://www.omg.org/spec/ReqIF

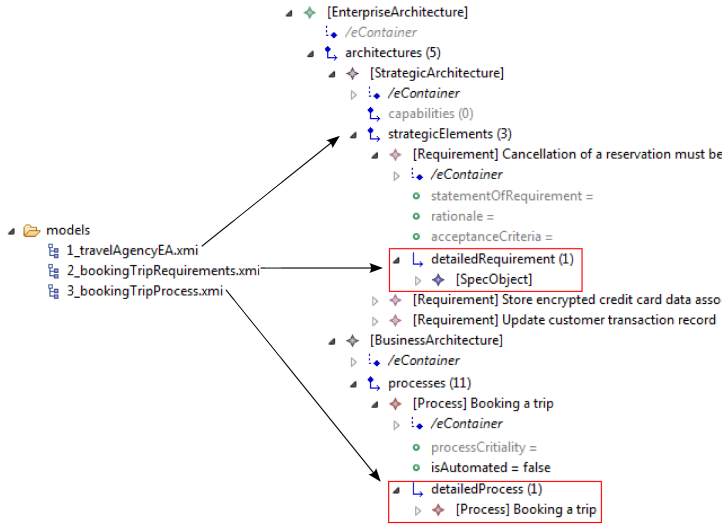

Fig. 3 Example of a view combining a TOGAF base model with ReqIF and BPMN ones, according to the viewtype definition from Figure 2 (excerpt).

attributes and references from the metamodels (those that are shown in Figure 2); and (*iii*) via *join*, the three metamodels are now linked through the new *detailedRequirement* and *detailedProcess* references.

Figure 3 presents a given view on a given Travel Agency example. This example plays the role of the system to be observed from a particular EA viewpoint in our present case. The presented view has been obtained using the previously specified viewtype that relies on the TOGAF, ReqIF and BPMN base metamodels. In this view, the (TOGAF) EA base model has been connected to both a (ReqIF) requirement base model and a (BPMN) business process base model. The *StrategicArchitecture* element has an enriched catalog of *strategicElements* composed of requirements that are now associated to their extended definitions using the ReqIF standard (via the *detailedRequirement* view association to elements from the requirement model). Similarly, the *BusinessArchitecture* element enumerates processes that are now directly connected to their equivalents from the business process model (via the *detailedProcess* view association). Moreover, only the attributes selected in the viewtype definition are being shown, such as *processCriticality* and *isAutomated* of class *Process* from the TOGAF metamodel.

3 Survey Method

This section explains the method we followed for realizing this survey. In particular, we describe how we extracted the required data in order to be able to build a relevant selection of approaches for views/viewpoints on models.

3.1 Approach Selection Process

The approach selection process was conducted in three main steps. First, we started selecting approaches based on our own experience about working on model view approaches during the last past years, e.g., [15, 17]. This very first selection contained 10 distinct approaches we were personally aware of. Second, in order to validate and potentially complement our knowledge of the field, we decided to explore bibliographic data sources. To this intent, we followed the guidelines by Kitchenham and Charters [40] concerning literature search and selection strategy. Finally, we performed snowballing by double-checking the references used within the finally selected publications as well as papers citing the selected papers. The overall selection process is summarized in Figure 4 and detailed further in the following subsections.

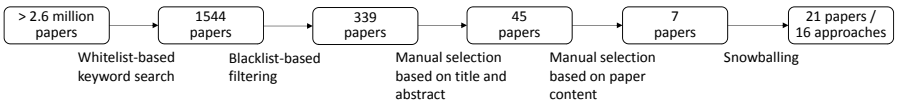

Fig. 4 Overview of the used selection process.

3.1.1 Data Source and Search Strategy

We used DBLP[6] as our core electronic database to search for primary studies on model view approaches. To avoid missing possibly relevant approaches, we decided not to put a specific period constraint for the search. Based on the topic of this survey, we defined the terms of the search query (according to the recommendations of Kitchenham and Charters [40]). Thus, we considered the terms "model" and "view" as the main constituents of our search query. We could have also included more search terms but most of them are already covered by the general term "view" (for instance the related term "viewpoint"). As we used a very general search query, we expected a huge result set requiring special treatment to select the most appropriate studies. This is explained later on as the pruning process. For automating the search process described here, we used the database dump of DBLP[7] along with SQL database query capabilities.

When executing the search query on this DBLP dump, we received a complete result set consisting of 1544 references. Based on this initial set of records, we then determined the set of relevant publication sources by a semi-automated pruning process.

[6] `http://www.dblp.org/search`
[7] `http://dblp.l3s.de/dblp++.php`

3.1.2 Pruning Process

In what follows (and as proposed by Kitchenham and Charters [40]), we describe the used inclusion and exclusion criteria. We also present the different pruning phases applied to select the final set of relevant papers, as found from the search results introduced before.

The selection of the primary studies was carried out based on the initial set of records we obtained from executing the search query against the DBLP data sources. To extract the studies relevant for this survey from the total number of studies, we passed through several pruning stages. In each one of these stages, the number of studies was significantly reduced compared to the result of the previous stage. At each stage, the following criteria were used to determine which approaches would move on to the next phase (and which ones would be ignored).

Inclusion and exclusion criteria Studies relevant for this survey must propose a view/viewpoint approach dedicated to models. Thus, they have to support viewtype definitions which interact with base metamodels as well as view definitions which interact with base models (cf. Section 2 for more details on the terminology used in this paper). From this overall principle, the inclusion and exclusion criteria that have been applied in this pruning process are the following ones:
Inclusion criteria

1. Proposed approaches that allow to define viewtypes and compute views conforming to them.
2. Proposed approaches that are suitable to work with models of arbitrary modeling languages.
3. Proposed approaches that are usable inside the modelware technical space, i.e., based on MOF or on any related metamodeling infrastructure (e.g., EMF/Ecore).
4. Proposed approaches that allow for user-defined viewtypes.

Exclusion criteria

1. Proposed approaches that are bound to a particular modeling language[8].
2. Proposed approaches that are not residing in the modelware technical space.
3. Proposed approaches that only offer a limited and fixed set of predefined viewtypes.

Pruning stage 1: Blacklist-based keyword search After the whitelist-based keyword search determining the initial set of studies, we conducted a blacklist-based keyword search. This stage was needed as we found out that many

[8] Note that comprehensive general-purpose modeling languages, such as UML, can be regarded as a composition of several domain specific languages.

papers which were part of the initial selection actually reside in the "computer vision" area. Thus, we have excluded papers having titles containing the following keywords: 3D, image, network, motion, video, multi-agent, face, spatial, computation, 3-D, multimedia, point, Markov, recognition, detection, Gauss, Bayes. Finally, we have also manually filtered conferences/journals related to the computer vision domain. This resulted in a set of 339 papers to be investigated during the pruning stage 2. Please note that the reduction to a human manageable number of studies at this stage was important. Indeed, the two remaining stages are hardly achievable automatically, and thus, were conducted manually.

Pruning stage 2: Manual selection based on title and abstract It is often rather difficult to determine the relevance of a study by considering its title only. Thus, we decided to evaluate in this stage both the title and abstract of each study against our inclusion and exclusion criteria. Generally, we acted in a conservative manner in this stage. In some cases, more information than the title and abstract is actually required to determine whether a study is relevant for this review. Especially, we filtered out approaches not directly applicable to models, i.e. approaches from different technical spaces than modelware (e.g., view approaches for XML data and ontologies). This gave us a reduced set of 45 papers to be investigated further in the final pruning stage.

Pruning stage 3: Manual selection based on detailed content In our final pruning stage, we carefully read the remaining papers considering both the survey's objective and the defined inclusion/exclusion criteria. After all, we selected 6 relevant approaches (actually corresponding to 7 different papers). Interestingly, the approaches finally selected after the whole pruning phase were all included in our initial list of known approaches. As a consequence, we are confident that this initial set of approaches already covers a large portion of published approaches for views/viewpoints on models.

3.1.3 Snowballing

Following the application of our methodology, we wanted to double-check that we did not miss any potentially relevant approaches. Indeed, some papers could have been missed during the search process for different reasons. For instance, some workshop papers are only indexed by ACM and some other papers have not yet been indexed by DBLP. Furthermore, some papers are not actually using the term *view* or similar in the title (e.g., if they present so-called composition or extension approaches). Finally, some existing solutions are tools, and thus, do not necessarily have indexed publications on DBLP.

As a consequence, based on the 7 selected relevant studies from our literature search, we performed backward and forward snowballing [68] by investigating the references they do cite as well as using Google Scholar[9] to search

[9] `http://scholar.google.com`

for papers citing the already found papers, respectively. This way, we have been able to identify some more approaches by applying the aforementioned inclusion/exclusion criteria to these extra-references as well.

The final result of the overall process (including this last snowballing stage) is presented in Table 1. This table provides, in addition to a representative name for each selected approach, the main publication(s) from which the data has been extracted out for this survey. At the end, we considered 16 different approaches as relevant for this survey.

3.2 Data Extraction Process

In order to study the selected approaches and to write down this paper accordingly, the data extraction process itself has been divided in different stages. It mainly concerns the terminology, feature model and the evaluation of the selected approaches for views on models.

First, initial results have been achieved by having a two-day face-to-face meeting in Vienna (Austria) between the four authors of this article. The objective was to:

1. Determine the initial set of approaches as well as the initial set of corresponding publications.
2. Agree on an overall terminology to be used all along the work.
3. Produce a first draft version of the feature model to be presented in this study.

During the following weeks, we also worked on establishing and applying the methodology described in this section.

Second, relying on this baseline, we distributed the selected approaches (i.e., both the ones from the initial set and from our systematic literature

Table 1 Selected approaches.

Approach	**References**
Cicchetti	[20]
EMF Facet	[25]
EMF Profiles	[44]
EMF Views	[14, 15]
Epsilon Decoration	[42]
Epsilon Merge	[41]
FacadeMetamodel	[57]
Kitalpha	[45]
ModelJoin	[16, 17]
OpenFlexo	[31]
OSM	[7]
Sirius	[63]
TGGvv	[36, 37]
TGGmv	[4]
VIATRA Viewers	[23]
VUML	[5, 56]

search) among us in order to individually evaluate them. After having performed all evaluations, we cross-checked several times the obtained results. This way, we also progressively revisited the terminology definition as well as the feature model in an iterative process.

Finally, we contacted the research community by sending the internally consolidated results to the actual authors of the investigated approaches. Thanks to this, we collected their direct feedback concerning the terminology, our proposed feature model and the evaluation of their respective approaches. This feedback, which we integrated in the final result of our survey, gives us more confidence on the relevance of the study we present in this paper.

4 A Feature Model for Characterizing Model View Approaches

Based on our own experiences working on/with model views in the past years, and on a deep study of the related state-of-the-art (cf. Section 3 for the used method), we propose in this paper a feature model (see Figure 5) describing what we consider to be the main characteristics of a model view mechanism. This feature model will be used in Section 5 to better describe and compare the model view approaches we have identified and selected.

We have identified three main categories of features that model view mechanisms can potentially cover. The first one gathers capabilities which concern the general *type structure* of the mechanism. The two others distinguish between *design time* and *runtime* aspects of the view/viewtype specification, computation and handling. We describe each one of these categories and the sub-features they contain in the following subsections.

4.1 Type Structure

- **Metamodel/Model Arity:** A fundamental aspect of a model view mechanism is its arity at both metamodel- and model-level. Some mechanisms allow specifying viewpoints/viewtypes over a single metamodel only, while others also allow combining several metamodels together in a same viewpoint/viewtype specification. Similarly, depending on the considered mechanism, corresponding views can be computed over a single model and/or combining several distinct ones. In the latter case, the involved models can potentially conform to different metamodels (as used in the related viewpoint/viewtype specification).
- **Closedness:** In addition to arity, we can differentiate two main families of mechanisms regarding their closedness. The first one limits the definition of views as subsets of the base model(s). The second one allows for views in which the content coming from the base model(s) can be augmented, e.g., with newly computed or manually entered data. Thus, the closedness is also related to the operations supported by the used viewtype and query languages (cf. the corresponding descriptions of these features). For

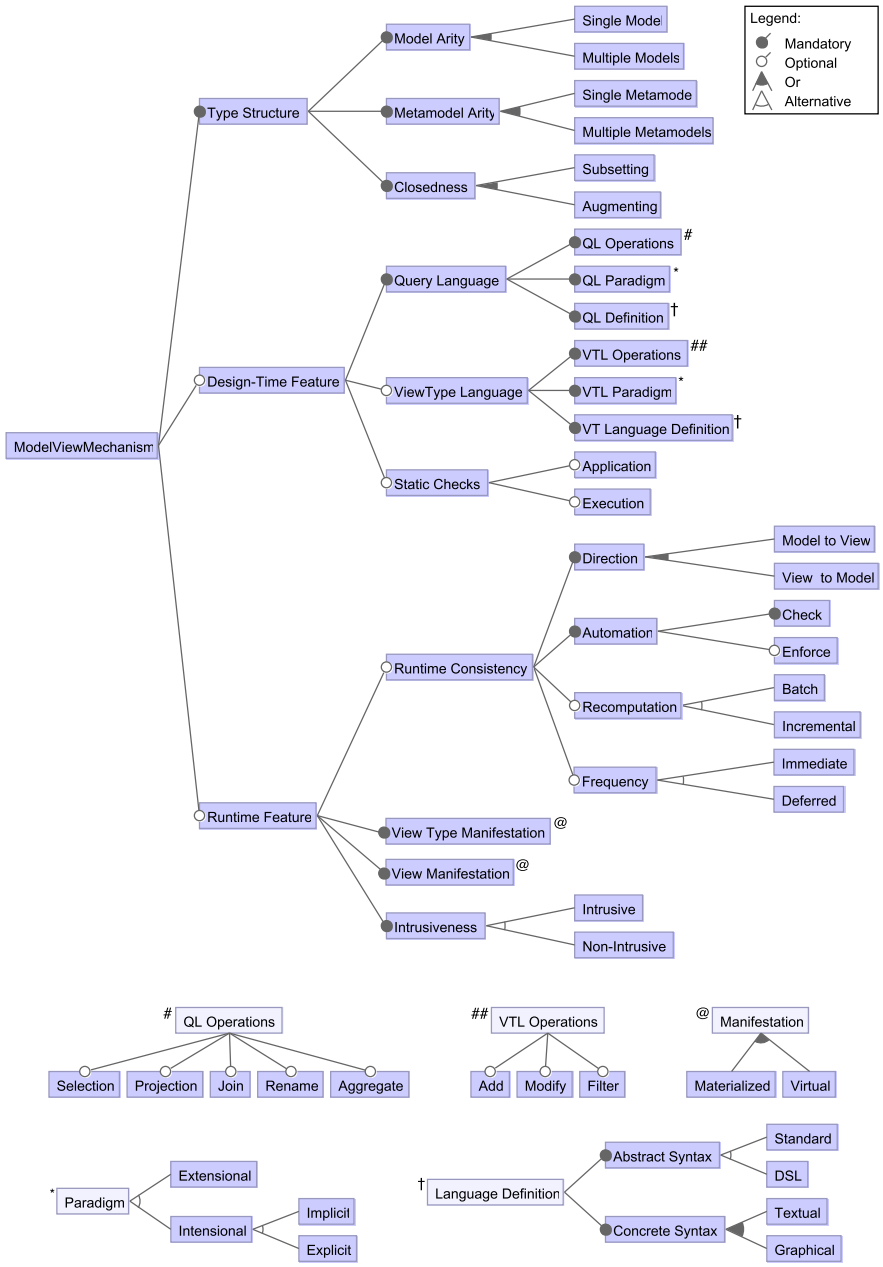

Fig. 5 A feature model for model view approaches.

instance a viewtype language can provide (or not) add operations, and a query language can come with some aggregate functions.

4.2 Design Time Features

As far as design time is concerned, we explicitly distinguish between two main kinds of languages (though they quite frequently appear mixed together in practice). The first kind of language is used to specify the viewpoints and build corresponding viewtypes, i.e., to specify the metamodel for a new viewtype. We call it the *View Type Language*. The second kind of language is used for corresponding computation at view-level, i.e., to populate the view with the expected data. We call it the *Query Language*. We describe the properties of these language kinds in the following.

- **Viewtype Language:** It has an abstract syntax that can be either based on an already existing standard and/or general-purpose modeling language (e.g., UML), or be implemented as a domain-specific modeling language (e.g., addressing more particularly a given set of concepts). As usual, its concrete syntax can be textual, graphical or eventually a mix of both. As said before, the viewtype definition itself can be directly connected to the used Query Language (cf. the explanation of this particular feature) or somehow independent from it.
  Any given viewtype language globally follows a particular paradigm for expressing viewpoints: definitions can be made either extensionally or intensionally (and explicitly or implicitly in the latter case). A definition is considered extensional when the actual element types to be part of the corresponding view are listed exhaustively. On the contrary, it is intensional when it rather provides the properties/conditions needed for inferring the corresponding view (elements). When the viewtype definition is intensional and explicit, it typically uses a combination of core add/modify/filter operations to be applied on the base metamodels. In the case of an intensional and implicit definition, some underlying operations or conditions can be applied by default/systematically.
- **Query Language:** This same extensional vs. intentional distinction also applies to the Query Language. Indeed, queries can be defined (extensionally) by providing a fixed set of expected values or elements. They can also use (intentionally) more complex expressions, explicitly stated or implicitly called, implying further computations in order to retrieve the targeted elements.
  In addition to that, and similarly to the viewtype language, a given query language can be based on a standard query language or equivalent (e.g., Structured Query Language (SQL) or Object Constraint Language (OCL)) or be a domain-specific query language targeting a particular domain or range of applications. The main capabilities of a given query language are provided by the operations it actually supports. Differently from the viewtype language, its operations focus on how the model-level data is

selected and manipulated to be integrated into the resulting view (in a way that conform to the viewtype definition). Therefore, a first set of operations consists in reducing the scope of the original metamodel(s) by selecting and/or projecting only some of the model elements in the produced view. A second set of operations is about complementing the view data by adding extra-information that can be derived from corresponding model elements (by join or aggregation) and/or manually added by a user (rename).

– **Static Checks:** Finally, different kinds of static checks can be performed at design time on the specified viewtypes and related queries. In some cases, they may concern the application of a given viewtype specification on the concerned base metamodel(s). In others, the pre-execution or parsing of the corresponding defined queries may be of interest. Two key properties of viewtypes arise here: (*i*) applicability (will the view computation ever return a non-empty set?) and (*ii*) executability (will the resulting view be ever consistent with the possible constraints defined at the viewtype-level?).

4.3 Runtime Features

From a runtime perspective, there are also different interesting aspects to consider for a model view mechanism. We describe them in the following.

– **Runtime Consistency:** There are several important consistency features to be considered when views are actually computed and handled. On one hand, synchronization can be supported in two main directions: from the model(s) to the view and from the view to the model(s). On the other hand, consistency verification can be more or less automated, following check or enforce strategies notably.
The way the recomputation is performed can also vary. It can be made incrementally by updating only the concerned elements in the view, e.g., when some elements from the base model(s) have been changed. It can also be realized in batch mode at given moment(s) in time, systematically recomputing the full content of the view in this case.
The point in time when the recomputation is performed is described by the Frequency feature. Immediate recomputation means that the synchronization is triggered directly after each change to a view (or to one of its base models). Deferred computation means that synchronization is performed at defined points in time, but can span several editing steps.
– **View/Viewtype Manifestation:** There are two manifestation dimensions to be possibly taken into account: a given mechanism can provide support at viewtype-level (i.e., metamodel-level), at view-level (i.e., model-level), or at both levels of course. In all cases, the results can be actually realized differently. They can be concretely materialized, e.g., with the creation/duplication of actually new (meta)model elements, or virtual in the sense of only relying on proxies to already existing (meta)model elements. The used type of manifestation can have a direct impact on the the way the

viewpoints/viewtypes and views are computed (cf. the description of the Runtime Consistency feature). For instance, synchronization can be made easier when using a virtualization approach instead of a duplication-based approach.
– **Intrusiveness:** Finally, a model view mechanism can also have different intrusiveness strategies, depending on whether or not it actually allows the alteration of the concerned base metamodels and models. Indeed some approaches do not modify the original (meta)models that can thus continue to have their own living, while some others do and directly impact the actual (meta)models content.

5 Description of Selected Model View Approaches

As previously explained in Section 3, we selected a number of model view approaches which are particularly relevant in the context of our study. They are used as our basis for presenting the current state-of-the-art in this area. We describe each one of these approaches and summarize their main characteristics in what follows. To realize this, we notably relied on the terminology and feature model we already introduced in this paper. Table 2 summarizes the overall results of our detailed study.

Cicchetti. The hybrid multi-view modelling approach by Cicchetti et al. [20] can be used to create sub-metamodels of existing Ecore metamodels and use them as customized view types. It is a hybrid approach in the sense that it uses the base models for the synchronization of the views (i.e., a *projective* approach [1]) and offers the creation of stand-alone models as views (i.e., a *synthetic approach* [1]).

The sub-metamodels, which are to be used as view types, are defined using a wizard in the Eclipse development platform. These view types have to be consistent with the base metamodel in such a way that instances of the view types, i.e., the views, are also valid instances of the original metamodel. Thus, the approach is strictly closed. Synchronization mechanisms for maintaing consistency among the views are then generated automatically using a combination of Eclipse-based technologies, such as ATL, Acceleo, Epsilon, and QVT-O. The mechanisms are based on model differencing. A difference metamodel is created automatically from the base metamodel and the view-type metamodel. From these difference metamodels, model transformations are generated that synchronize the views with the models.

Finally, an Eclipse plug-in is generated automatically, which provides an editor for the creation and manipulation of the views.

EMF Facet. EMF Facet [25] is an open source Eclipse/EMF-based tool, initially developed in the context of MoDisco [13] and then externalized. It provides a generic lightweight extension mechanism for existing EMF models. In particular, it allows to define so-called *facets* on given metamodels (i.e., Ecore

				Cicchetti	EMF Facet	EMF Profiles	EMF Views	Epsilon Merge	Epsilon Decoration	FacadeMetamodel	Kitalpha	ModelJoin	OpenFlexo	OSM	Sirius	TGGmv	TGGvv	VIATRA Viewers	VUML
Type Structure																			
Model Arity	Single Model			✓	✓	✓			✓	✓				✓		✓	✓		
	Multiple Models						✓	✓			✓	✓	✓		✓			✓	✓
MM Arity	Single MM			✓	✓	✓			✓	✓				✓		✓	✓		✓
	Multiple MMs						✓	✓			✓	✓	✓		✓			✓	
Closedness	Subsetting			✓			✓	✓			✓	✓	✓	✓	✓	✓	✓	✓	✓
	Augmenting				✓	✓	✓		✓	✓	✓	✓			✓	✓	✓	✓	
Design Time Features														∅					
Query Language	QL Operations	Selecting		✓	✓		✓				✓	✓	✓		✓	✓	✓	✓	✓
		Projecting		✓	✓		✓				✓	✓	✓		✓	✓	✓	✓	✓
		Joining					✓	✓			✓	✓	✓		✓	✓	✓	✓	✓
		Rename					✓	✓			✓	✓	✓		✓	✓	✓	✓	✓
		Aggregation			✓		✓	✓			✓	✓	✓		✓	✓	✓	✓	
	QL Paradigm	Extensional			✓														✓
		Intensional	Implicit							✓									
			Explicit	✓	✓		✓	✓			✓	✓	✓		✓	✓	✓	✓	
	QL Definition	Abs. Syntax	Standard		✓										✓				✓
			DSL	✓			✓	✓			✓	✓	✓		✓	✓	✓	✓	
		Con. Syntax	Textual		✓		✓	✓			✓	✓	✓		✓			✓	✓
			Graphical	w									✓			✓	✓		
VT Language	VTL Operations	Add			✓	✓	✓		✓		✓		✓			✓	✓	✓	
		Modify					✓				✓	✓	✓						
		Filter					✓				✓	✓	✓						
	VTL Paradigm	Extensional		✓	✓	✓	✓	✓	✓		✓		✓		✓	✓	✓	✓	✓
		Intensional	Implicit																
			Explicit							✓		✓							
	VTL Definition	Abs. Syntax	Standard	✓		✓						✓				✓	✓		✓
			DSL		✓		✓	✓	✓	✓	✓		✓		✓			✓	
		Con. Syntax	Textual				✓	✓			✓	✓	✓					✓	
			Graphical	w	✓	✓			✓	✓			✓		✓	✓	✓		✓
Static Checks	Application					✓			✓	✓		✓	✓						
	Execution			✓			✓	✓		✓	✓		✓		✓	✓	✓	✓	✓
Runtime Features																			
Consistency	Direction	Model→View		✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓
		View→Model				✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Automation	Check		✓								✓		✓					✓
		Enforce				✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
	Recomputation	Batch			✓		✓	✓			✓	✓			✓	✓	✓		✓
		Incremental		✓		✓			✓	✓			✓	✓		✓		✓	
	Frequency	Immediate		✓		✓	✓		✓	✓	✓		✓	✓	✓	✓	✓	✓	
		Deferred			✓			✓				✓							✓
VT Manifest.	Materialized			✓		∅		✓	∅	✓	✓	✓	✓	✓		✓	✓	✓	✓
	Virtual				✓	∅	✓		∅						✓				
View Manifest.	Materialized			✓		✓		✓	✓		✓	✓			✓	✓			✓
	Virtual				✓		✓			✓			✓	✓			✓	✓	
Intrusiveness	Intrusive													✓			✓		✓
	Non-Intrusive			✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓		✓	

Table 2 A comparison of existing model view approaches (✓=feature supported,∅=not applicable,w=Wizard).

models) in order to complement them with new computed types (concepts), attributes and references (properties). Modification and filtering of existing elements are not possible in the current version. The definition of a facet is stored in a specific and separate Facet model. If available for a given metamodel, a facet can then be applied on its corresponding models (e.g., within a model browser) in order to dynamically extend them at runtime. A facet can thus be seen as a view type that is specified over a single metamodel with the objective to extend it with newly computed information. When applied to given models, it allows to obtain corresponding views on them (i.e., extended models).

A core aspect of the approach is that it relies on a query abstraction framework that gives the possibility to plug any type of query (e.g., Java, OCL) to the facet so that the newly added elements can be computed automatically. In

addition, another specificity of the approach is that everything is performed at runtime, meaning that the "extended" model (i.e., a view) is dynamically computed when the facet is applied on. The original model is not modified and the elements added in the view are only stored "virtually" in memory until the facet is unloaded or the model is closed.

EMF Profiles. EMF Profiles [44] is an approach and corresponding Eclipse/EMF-based prototype that provides another generic lightweight extension mechanism for EMF models. Directly inspired from what is already widely used in the UML world, it basically generalizes the concept of profiles to be used with all possible (meta)models. Thus, instead of allowing to extend only the UML metamodel, profiles can be defined on top of any Ecore model (i.e., any metamodel) and then applied on any corresponding instance (i.e., any corresponding model). A profile specifies a set of named stereotypes applicable only to selected types of elements and that come with their own additional properties. A profile can thus be seen as a viewpoint that is specified over a single metamodel with the objective to extend it with annotations on existing elements. When applied to given models, it allows to obtain corresponding views on them (i.e., extended models).

A core aspect of the approach is that it is intended to be used in a semi-automatic manner: a profile application is not computed automatically by default, instead the user triggers the application of stereotypes and many times the additional information stored in the view is added directly by the user. A given profile application is persisted as a separate model in addition to the base model (in contrast to UML profiles). Thus, it can also be potentially computed by a model transformation if required and/or relevant in particular cases. Another interesting aspect of the approach is its capacity to define meta-profiles, i.e., profiles that are defined at meta-metamodel level. Thus, they are applicable on all models (independently from the metamodel they conform to) and allow to represent viewpoints which are directly reusable for different modeling languages.

EMF Views. EMF Views [15] is an approach and corresponding Eclipse/EMF-based prototype that provides capabilities for specifying and obtaining views on top of models which potentially conform to different metamodels. Inspired from views in databases and the well-known SELECT-PROJECT-JOIN operators in relational algebra, it comes with an SQL-like DSL for defining viewpoints at metamodel-level (i.e., on one or several metamodels). Such viewpoints are then used to generate automatically corresponding views at model-level (i.e., on one or several models). Like views in databases, the obtained model views are mainly read-only in the current version, as only a limited set of modifications can be propagated back to the underlying models.

The core aspect of the approach is that it relies on a same generic model virtualization framework for realizing both viewpoints and views. Thus, a viewpoint is actually a virtual metamodel that only contains proxies to elements from the original metamodels. Similarly, a view is a virtual model that

only contains proxies to elements from the original models. Additional links between/to elements from different metamodels/models are stored in a separate weaving model. This offers interesting properties to EMF Views in terms of transparency (a view is seen and handled as a regular model), synchronization (a view always reflects the content of the base models), non-intrusiveness (a view is not altering the base models) and scalability (a view does not duplicate any actual model element). In addition, this also allows EMF Views and its model virtualization back-end to be reused in different contexts, such as lightweight metamodel extension for instance [14].

Epsilon Merge. There is a first Epsilon-based approach [41] which provides a hybrid rule-based language (i.e., declarative rules with imperative bodies) for combining both homogeneous and heterogeneous models. As a merging approach, it allows to produce a new (merged) model out of several original models. Such a merged model can be considered as a view gathering elements coming from several different models, the merge specification in the Epsilon Merging Language (a so-called EML Module) thus acting as the viewpoint definition (of the merged metamodel) in this case. The overall merge process and its four consecutive steps (comparison, conformance checking, merging, reconciliation/restructuring) are actually realized thanks to the combined used of several Epsilon languages (respectively the Epsilon Comparison Language (ECL), the Epsilon Validation Language (EVL) and EML for the last two steps).

A core aspect of this approach according to our present study is that it is merge-based. Thus, it covers only partially the set of capabilities generally expected when dealing with model views. For instance, basic synchronization between the original models and the generated merged model (i.e., the view) is not provided by default (even if some support can be implemented based on the merge trace model produced during the merge process).

Epsilon Decoration. There is a second Epsilon-based approach [42] that introduces a model decoration support enabling the annotation of models with additional information that is not necessarily supported by existing metamodel(s). A decorated model can be considered as a view on the original model complemented with some new manually added information. In the proposed solution, which differs from most other approaches, there is no explicit viewpoint definition and decorations are simply represented in a generic way, namely as tag/value pairs. Interestingly, the approach also proposes a bidirectional support for covering both automated decoration extraction from a previously annotated model (via its diagram) and decoration injection (or application) to a non-annotated model. This model decoration approach is implemented on top of the Epsilon framework, notably by reusing its Epsilon Model Connectivity (EMC) layer. The idea of having such a layer providing generic model loading, storing, querying and updating capabilities has been be directly inspired by existing relational database connectivity layers.

A core aspect of the approach is that it has the following two main characteristics: (*i*) non-intrusive as the original model is not modified and decorations are stored in a separate decorator model, and (*ii*) transparent from a usage perspective as decorations can be handled manually and programmatically as if they were parts of the original model. However, the trade-off is that specific decoration injection and/or extraction transformations have to be specified for each decoration alternative (i.e., expressing a particular viewpoint).

FacadeMetamodel. The FacadeMetamodel [57] benefits from the realization that most domain-specific languages are often just subsets of UML itself. It presents an approach to build new DSMLs by generating a kind of "facade" that looks like a pure DSL to the modeler. However, it is in reality using UML in the back thus allowing full reuse of the UML infrastructure.

Given the extensibility and ease of use of this approach, the FacadeMetamodel may be used to build views on top of UML models by aliasing (renaming), refining, pruning and extending (using the UML profile mechanism) the UML language. The approach takes this customization model and generates an Ecore metamodel representing the new language/viewtype that can then be used to generate a custom user interface (e.g., in Papyrus[10]).

The new language appears as a complete pure DSL to end-users. Therefore, all modeling tools can use it transparently and it is automatically populated from the base UML models. Updates on the DSL level are propagated back to UML and stored as normal UML models. Updating those same models with standard UML tools is not recommended due to the possibility of inconsistent modifications.

Kitalpha. In the context of the PolarSys project[11], Kitalpha[12] is proposed as a framework to define model views inspired from the use of viewpoints in architectural modeling. Thus, the notion of view is very general in Kitalpha and covers the following points: abstract syntax, notations (such as icons), concrete syntax (textual and graphical), rules (e.g., check, transformation), services and tools. Furthermore, Kitalpha focuses also on reusing views by providing inheritance and aggregation for viewpoints.

A textual domain-specific language is provided to define these general building blocks of a viewpoint in Kitalpha. In addition, dedicated DSLs are provided to define the internals of the different building blocks. For instance, a language for defining the abstract syntax of viewpoints is available. With this language, metaclasses from base metamodels may be extended with more specific classes, new classes may be introduced, and additional references may be added as well.

[10] `https://eclipse.org/papyrus`
[11] `http://www.polarsys.org`
[12] `http://www.eclipse.org/proposals/polarsys.kitalpha`

ModelJoin. ModelJoin [16, 17] is a domain-specific language and tool for the creation of views on heterogeneous models (i.e., models that are instances of different metamodels). ModelJoin is a declarative language with a human-readable textual concrete syntax that bears similarities to that of SQL. The language is used to describe the desired properties of a view. It abstracts from the technical details of how views are created.

In relational databases, the table schema of the result of a query is dependent on the columns chosen by the projection operators. Transferred to the modeling, this means that such a query not only defines the elements in a view, but also the kinds of elements which can be in a view, i.e., the metamodel of a view (called *view type* according to the terminology of [30]). Thus, the implementation of ModelJoin generates both a target metamodel and transformations that create target models from the input (base) models.

ModelJoin has recently been extended to support editability inside the views [18]. For this purpose, OCL constraints that limit view editability are generated together with the views. These constraints define the possible edit operations for which the resulting views can be translated back to a valid base model. If no such translation is possible, the view may be adapted automatically with an automatic fix so that the resulting view is translatable. The prototypical implementation is based on Eclipse technologies such as Xtext and Xtend, and uses QVT-O as the target language into which the transformations for the synchronization between base models and view are generated.

OpenFlexo. OpenFlexo [31] is a generic solution allowing to assemble and relate, without duplication, data coming from various kinds of data sources. Its main goal is to support the federation of data from heterogenous technical spaces (EMF, XML, OWL, Excel, etc.) into the same conceptual space realized as a kind of virtual view. It comes with several components, including notably the Viewpoint Modeler and ViewEditor. The former is for specifying viewpoints that indicate how to mix together the different types of data. The latter intends to provide regular view visualization and editing capabilities, depending on previously specified viewpoints. To integrate these components, the solution also comes with an underlying model federation framework. This allows for homogeneous handling of data as models.

A core aspect of the approach is that it also provides the ability to define synchronized views on models, e.g., stored as EMF models. As soon as a virtual view is computed (from potentially multiple different models), the view is connected with the different base models. The other way round, there is a mechanism for indirectly connecting the different base models to the view. For instance, this allows propagating changes from one base model to other base model(s) via the common view. Furthermore, changes on the view elements can be propagated back to the corresponding base models.

OSM. The Orthographic Software Modeling (OSM) approach [7] aims at establishing views as first-class entities of the software engineering process. In

the envisioned view-centric development process, all information about a system is represented in a single underlying model (SUM). Even source code is treated as a special textual view. The OSM concept is based on three main principles: dynamic view generation, dimension-based view navigation, and view-oriented methods.

User-specific custom views are generated dynamically based on transformations from and to the SUM. These views are organised in independent (orthogonal) dimensions. Technically, a view is a model of its own, which also has a metamodel. Model-to-model transformations allow to create the views dynamically from the SUM. However, this requires that bi-directional transformations exist for every view type. Such transformations can provide the synchronisation of the views with the SUM. In addition, edit operations can be propagated back to the SUM likewise. The complexity of a hub-and-spoke architecture like OSM is linear in terms of the number of transformations that have to be written and maintained. This notably contrasts with the quadratic number of transformations in a peer-to-peer synchronisation scenario for views.

OSM also encompasses a development process with a *developer* role, who uses the generated views, and a role called *methodologist*, who creates the different view types along the orthogonal dimensions. Atkinson et al. [6] have developed a prototypical implementation of the OSM approach based on KobrA. It relies on UML and OCL, and offers the dimensions abstraction (defined by notions of model-driven development), variability (defined by notions of product line engineering), compositionality (defined by notions of component-based development), encapsulation (e.g., public/private) and projection (structural/operational/behavioural).

Sirius. Sirius [63] is an open source Eclipse/EMF based tool that is intended to facilitate and speed up the construction of industrial graphical modeling solutions. Considering the domain metamodels which are key assets inside companies, and thus, cannot be changed or modified easily, it allows to specify different concrete representations on top of them expressing different viewpoints (for different stakeholders). Within Sirius, a given Viewpoint Specification Model (VSM) is defining a set of logically organized representations which are graphical constructions to represent the actual data (i.e., the original model). Such representations can be diagrams, tables, matrices or trees. A viewpoint specification also describes how the different elements from the original metamodel(s) should be actually mapped to the various representations. These mappings can go from basic ones (e.g., one concept/one box in a diagram) to complex ones (resulting from complex query computations, e.g., defined in the Acceleo Query Language (AQL)). Thus, applying such a viewpoint, different graphical views on the same model(s) can be provided to different types of users/stakeholders.

A core aspect of the approach is the clear separation between the metamodel that is conceptual and the viewpoint that is considered as a purely representation asset (mostly in the sense of graphical representation). Another aspect is that the provided views can be made completely editable if

needed, the required editing capabilities being also specified within the Viewpoint Specification Model. In this respect, Obeo Designer Team (as a commercial extension of Sirius) comes with more powerful additional features dealing with collaborative editing of such views and the management of related conflicts.

TGG-based approaches. Triple Graph Grammars (TGGs) [61] have been proposed as a means of specifying a consistency relation between two graph languages. TGGs are a well-studied formalism to define bi-directional transformations including not only the back and forth translation of models, but also the comparison of models and their synchronization. Thus, they are also applicable as a base mechanism to solve model view problems. In particular, two TGG-based approaches have been presented in the past for defining views. Both TGGvv (virtualized view) and TGGmv (materialized view) are supported by the EMF-based, bootstrapped eMoflon model synchronization tool [46]. In the following, we summarize both approaches.

The first approach of using TGGs for model views (TGGvv) has been presented by Jakob et al. [36, 37]. In this approach, the base metamodel is aligned with the viewtype by defining a correspondence model between them. Based on these three models, models conforming to the base metamodel can be filtered to yield virtual views conforming to the viewtype. The advantage of this approach is that the views are virtual in the sense that elements of the base model are simply interpreted as elements of a view. This simplifies synchronisation tasks in many cases. The approach, however, requires nontrivial changes to the base metamodels and has not been fully formalised for complex metamodels or multiple views on the same base metamodel.

TGGvv has been complemented by a recent TGG-based approach for materialized views (TGGmv) [4]. It notably allows for separate view models without requiring any changes to the base metamodels. This approach provides a formalisation of View TGGs, as a restriction of existing TGG theory to the special asymmetric case of view specification (as TGGs are symmetric in general). It shows that the chosen restrictions can be suitably exploited to enable highly efficient view synchronisation.

VIATRA Viewers. The VIATRA Viewers[13] approach emerged from EMF IncQuery [65] which is a framework for performing incremental model querying based on the RETE algorithm. The main advantage of EMF IncQuery, compared to many other model query approaches, is that queries do not have to be entirely evaluated in case model changes happen (but only queries which are actually concerned by the changes). Of course, this capability plays also an important role for model view approaches. Thus, EMF IncQuery can be extended for dealing with model views [23]. In particular, the EMF IncQuery language is extended by so-called derivation rules which are defined with annotations to EMF IncQuery query patterns. Via the annotation of derivation

[13] `https://wiki.eclipse.org/VIATRA/Addon/VIATRA_Viewers`

rules, the viewtypes are defined and the query patterns are used to populate the views by simply reusing the EMF IncQuery support.

In addition to the views, trace models are computed between the views and the base models. These trace models are used to reason about changes of query pattern matches which allows to build synchronization support. In particular, if changes to the base models are performed, the query pattern matches are changed. This stimulates incremental updates in the view models. In recent work, the propagation of view change to the base models is also considered [62]. Finally, it has to be mentioned that VIATRA Viewers allows to build chain views, i.e., to build views on views.

VUML. VUML [5,56] is a UML profile that supports view-based modeling for the UML family of modeling languages. The proposed methodology includes the definition of *actors*, which all possess a unique viewpoint, and a design process that supports the actor concept. The design process is aligned with the OMG MDA process [52]. The viewpoint models which are tied to these viewpoints are UML class diagrams. They are composed into a VUML system model. The composition algorithms have been implemented in a composition metamodel and transformations in ATL.

The application of a viewpoint on a specific system is called a view. VUML provides the concept of *Multiviews Components*, which consist of both default base views (that can be used by all actors) and actor-specific views (that are connected to the base view via an extension relation). The actor concept can be used to structure the views and to implement access control. The semantics of VUML are described by a metamodel and textual descriptions. The views of a MultiView Component can have dependencies between each other, which are expressed with OCL constraints.

6 Research Agenda

Several interesting findings can be made from the aggregated Table 2 resulting from our previous evaluation of different model view solutions. There are some commonly shared aspects: for instance, all evaluated approaches require an explicit definition of the viewtype, and do not offer the possibility of deriving it on the fly when computing the view. Also, for each feature there is at least one approach covering it, showing the variety of existing solutions. Still, most of these solutions focus on a reduced set of features. We are still missing more general solutions that can be applicable in a broad number of scenarios.

By looking at Table 2, we can also see some features that seem to be more challenging since very few approaches provide some support from them. A typical missing feature is the verification support for viewpoint/view definitions. Most approaches do not support designers in the specification of the viewpoints and underlying viewtypes. For instance, they do not alert the users regarding the applicability or executability of these definitions to build actual views. The few that somehow ensure these properties are doing it more as a

side-effect, because the underlying view mechanism itself is strict enough to prevent from some possible issues. Moreover, graphical languages to express queries are hardly used, while they could be useful to allow less technical users defining their own viewpoints/views (as tools like graphical query builders for databases have proven to be).

Beyond these mentioned features, we would like to highlight a few more research challenges worth to be investigated in the coming years. Some of them are actually well-known recurring problems in any technical space where views are used in practice (e.g., in the database domain). Some others are more specifically related to our modeling context and have been identified by studying deeper the content of Table 2.

- **Terminology inconsistencies**. A different vocabulary is employed within papers from the literature we studied for this review. For instance, Melnik et al. [53] define model composition as the combination of two successive mappings into one. For Chechik et al. [19] and others, model composition refers to a specific model integration scenario where models with running interacting features are assembled. This also causes comparisons and building on top of existing approaches to become more difficult. This is especially evident if we look at papers targeting different modeling levels in the four-layer modeling infrastructure of OMG. Even papers that use the same type of techniques and share a similar conceptual goal may use a very different terminology (e.g., DSL combination vs model merging). This makes them quite often ignorant of each other. The present paper is an initial step in the direction of trying to solve this problem as far as views/viewpoints on models are concerned.
- **View updating problem**. In the general case, fully updating a view is not always possible [51]. This will notably depend on the kind of modifications applied, and on the operators used to compute the view. Indeed, some combinations may not result in a deterministic translation of the view update into a set of modifications on the base model elements. As a simple example, imagine a model view that displays the average value of a certain attribute from a given class. An update of this average value does not have a single way to be propagated back to the individual values from the base models: Should we proportionally increase all of them? Or rather assign the whole increment to a single one? A pragmatic solution would be to provide a uniform way to support several different model view update strategies. For instance, a listener on the model view could capture the update events and deals with them according to the instructions provided by the designer (e.g., similarly to the concept of INSTEAD-OF triggers as available in SQL). As we have observed, current solutions follow a more conservative approach where they basically restrict the changes as soon as they become complex to handle.
- **Incremental view maintenance**. For those approaches where (some of) the model view elements are automatically computed, a major problem is to incrementally update them after changes on the base models. As

seen in Table 2, current solutions typically ignore or provide very limited support to this feature. Always completely recomputing the whole view may be too costly and/or trigger undesirable side-effects in some cases. As introduced in the previous item, specific view update strategies could be implemented to provide the needed incremental support to deal with such scenarios. As a possible technical solution, they could rely on incremental model transformation techniques [8, 39]. However, there are some existing limitations depending on the complexity of the model view. For instance, non-distributive aggregate functions are very difficult to deal with [58].

- **Concrete syntax generation**. The (direct or indirect) definition of the abstract syntax of the view type is a key element in most approaches. Nevertheless, most of them do not offer explicit support to specify the concrete syntax part (as seen from Table 2). To make model views more easily usable by end-users, we should be able to display the view content graphically (and not just show it using default tree-like browsers). In order to achieve this, we could generate a default concrete syntax based on the concrete syntax(es) associated to the base metamodels/languages. We could also manually build one explicitly for a given viewtype. The graphical-oriented approaches proposed by solutions such as Sirius or Kitalpha are good examples to follow regarding these aspects.
- **Security aspects**. Views are typically used as a security mechanism to prevent people from accessing data they are not authorized to see and/or modify. This requires the availability of an access-control mechanism that enables designers to give read/write permissions (on specific model views) to particular categories of persons. Such a mechanism notably allows preventing them from accessing directly to the base models when not appropriate. As observed during our study, this is in general a green area for the modeling community. However, it makes sense to tackle related challenges once the specification of views/viewpoints on models comes into play. For instance, many approaches provide profiles or DSLs to annotate models with security characteristics of the system being modeled. Until now, they do not allow assigning explicit permissions to the model access itself. This could be possibly addressed by the use of model views in such contexts.

7 Related Work

In this paper, we focus on surveying approaches for views on models as described in Section 2 by following a feature-based categorization methodology. Besides this kind of solutions, several other related lines of research exist as well. They are related to views in the very general sense (as technical prerequisites), or propose a different interpretation of views for models. In the following, we first outline some of these research directions. Then, we enumerate and discuss other (feature-based) surveys in the field of model-driven engineering. Finally, we overview related surveys on the topic of views in technical spaces different from modelware.

The problem of view modeling is related to several other model management tasks in the modeling area. In some works [9, 53], the following core model management operators are defined: *match*, *compose*, *merge*, *extract* and *diff*. Of course, some of these operators may be built-in and combined into more general solutions for view modeling and view generation. Other additional operators have been subsequently defined, such as for the union and subsetting of models [2].

There are multi-viewpoint modeling solutions that provide model view capabilities such as ODP [48], UML [12] and Model Driven Web Engineering (MDWE) [55] approaches. They have in common that they all base themselves on a fixed number of viewpoints to decompose the systems they are representing. For many of the proposed languages, view modeling approaches have also been introduced. However, they are usually tailored language-specific solutions and cannot be applied on other languages.

The ISO 42010 standard [1] defines a conceptual model for software architectures. It gives a broad definition of the terms *architecture view* and *architecture viewpoint*, which deviate from the definition used in this paper. It distinguishes between two fundamental approaches to view-based modelling: A *synthetic* approach is when architects create views of a system and integrate them using model correspondences. A *projective* approach is when architects derive views from one or several base models using a (possibly automated) procedure. Several architectural standards claim conformance to ISO 42010, but many of them share different notions of the concepts defined in it [34].

Several papers focus on the problem of composing/combining domain-specific modeling languages (DSMLs). One of their main objectives is to achieve a global and unified modeling language to express system models from a set of fixed viewpoints. In this case, each one of these viewpoints is specified using a different DSML. In general, these approaches do not focus on the migration of the individual model views. In case they do, models are typically juxtaposed to conform to the abstract syntax of the composed DSML. Among the works focusing on this topic from a variety of backgrounds, we refer the interested reader to the areas of unification (also includes a review of previous works on the topic) [66], globalization of modeling languages [21], grammar-based composition [43], metamodel weaving [11], metamodel merging [71] and language embedding [35].

Other works deal with similar issues but at model-level. Thus, the view merging problem is typically applied to the challenge of building a single consistent and unified model view over a system. This is realized from a set of views specified independently by different groups of people. Many times, individual views are quite homogeneous and conform to the same language. However, their union is not trivial as it may induce (many) possible inconsistencies in the global view. Depending on the complexity of the involved views, dedicated algorithms may be required, e.g., for state machines [19] or block-based modeling languages [49]. More general approaches allow to express desirable properties on such a unified view [59], and even to merge views that may be inconsistent themselves in the first place [60]. In this respect,

Engels et al. [26] present an approach which considers also the operations (expressed as graph transformations) to be used in single views as well as during their integration.

Incremental model query and transformation approaches [27] are also related to model views, at least in the sense that they can act as a prerequisite for realizing efficient implementations of model view approaches This is also reflected in our feature model (cf. recomputation feature). Incremental techniques, as the name suggests, do not transform the complete input model to a new result each time. Instead, they only consider the differences between the current input model and the input model used in the last execution run to minimize the changes to be applied on the existing result. Several incremental approaches have been proposed in the last past years [10,38,65], but there are also other approaches published earlier [27].

Concerning our feature-based methodology for discussing different model view approaches, we follow a best-practice in publishing in the model-driven engineering domain. In previous works, several feature-based surveys have been already published [3, 22, 32]. In addition, related surveys are available which discuss prerequisites of model view approaches such as model transformations in general [54] or particular model transformation approaches [33]. However, to the best of our knowledge, we are not aware of any survey about model view approaches (except a preliminary work on a tool-oriented taxonomy for combining domain-specific languages with view-based modeling approaches [30]). In particular, current language workbenches have been evaluated regarding the specification of views for domain-specific models [30]. Our work is orthogonal in the sense that we are interested in dedicated model view approaches which may build on language workbenches (such as EMF). However, we do share small parts of definitions between our feature model and the taxonomy presented in this evaluation [30].

8 Conclusions

All along this paper, we have presented a recollection of both research and industrial development dealing with the important problem of views/viewpoints on models and the corresponding technological support. We do believe this research line is becoming more and more crucial for the modeling community, and also from a general software engineering perspective. This is especially true given the increasing complexity of industrial systems that need to be modeled, e.g., consider emerging cyber-physical systems and large-scale IoT systems.

We have motivated why we think many future applications of modeling will depend, one way or the other, on some kind of mechanism for dealing with views/viewpoints on models. We have also described what are the solutions available nowadays and how they currently compare to each other. To this intent, we categorized them according to a detailed feature model that aims to shed more light on this topic. This collaborative work has notably required

to face terminology problems that quite often hamper the combination and reuse of existing initiatives.

As further work, we plan to tackle some of the issues described in the proposed research agenda. This is expected to be performed mostly in relation with some of our ongoing and future collaborative projects on model-based technologies and their application in different industrial domains. We also hope this paper triggers a broader discussion within the community. This should allow enriching our current feature model and extending the approach comparison in the future. But, more than anything, this should raise the awareness on the research challenges around views/viewpoints on models and their practical usages.

References

1. ISO/IEC/IEEE Std 42010:2011 – Systems and software engineering – Architecture description. Los Alamitos, CA: IEEE (2011)
2. Alanen, M., Porres, I.: A Metamodeling Language Supporting Subset and Union Properties. Software and System Modeling **7**(1), 103–124 (2008)
3. Altmanninger, K., Seidl, M., Wimmer, M.: A Survey on Model Versioning Approaches. International Journal of Web Information Systems **5**(3), 271–304 (2009)
4. Anjorin, A., Rose, S., Deckwerth, F., Schürr, A.: Efficient Model Synchronization with View Triple Graph Grammars. In: Proceedings of the 10th European Conference on Modelling Foundations and Applications (ECMFA), pp. 1–17. Springer (2014)
5. Anwar, A., Ebersold, S., Coulette, B., Nassar, M., Kriouile, A.: A Rule-Driven Approach for Composing Viewpoint-oriented Models. Journal of Object Technology **9**(2), 89–114 (2010)
6. Atkinson, C., Bostan, P., Brenner, D., Falcone, G., Gutheil, M., Hummel, O., Juhasz, M., Stoll, D.: Modeling Components and Component-Based Systems in KobrA. In: A. Rausch, R. Reussner, R. Mirandola, F. Plášil (eds.) The Common Component Modeling Example, pp. 54–84. Springer (2008)
7. Atkinson, C., Stoll, D., Bostan, P.: Orthographic Software Modeling: A Practical Approach to View-Based Development. In: Proceedings of the 4th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE), pp. 206–219. Springer (2010)
8. Bergmann, G., Ráth, I., Varró, G., Varró, D.: Change-driven Model Transformations - Change (in) the Rule to Rule the Change. Software and System Modeling **11**(3), 431–461 (2012)
9. Bernstein, P.A., Melnik, S.: Model Management 2.0: Manipulating Richer Mappings. In: Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data (SIGMOD), pp. 1–12. ACM (2007)
10. Beyhl, T., Blouin, D., Giese, H., Lambers, L.: On the Operationalization of Graph Queries with Generalized Discrimination Networks. In: Proceedings of the 9th International Conference on Graph Transformation (ICGT), pp. 170–186. Springer (2016)
11. Bézivin, J., Bouzitouna, S., Del Fabro, M.D., Gervais, M.P., Jouault, F., Kolovos, D., Kurtev, I., Paige, R.F.: A Canonical Scheme for Model Composition. In: Proceedings of the Second European Conference on Model Driven Architecture: Foundations and Applications (ECMDA-FA), pp. 346–360. Springer (2006)
12. Breu, R., Grosu, R., Huber, F., Rumpe, B., Schwerin, W.: Systems, Views and Models of UML. In: UML Workshop, pp. 93–108 (1997)
13. Bruneliere, H., Cabot, J., Dupé, G., Madiot, F.: MoDisco: A Model Driven Reverse Engineering Framework. Information and Software Technology **56**(8), 1012–1032 (2014)
14. Bruneliere, H., Garcia, J., Desfray, P., Khelladi, D.E., Hebig, R., Bendraou, R., Cabot, J.: On Lightweight Metamodel Extension to Support Modeling Tools Agility. In: Proceedings of the 11th European Conference on Modelling Foundations and Applications (ECMFA), pp. 62–74. Springer (2015)

15. Bruneliere, H., Perez, J.G., Wimmer, M., Cabot, J.: EMF Views: A View Mechanism for Integrating Heterogeneous Models. In: Proceedings of the 34th International Conference on Conceptual Modeling (ER), pp. 317–325. Springer (2015)
16. Burger, E., Henß, J., Kruse, S., Küster, M., Rentschler, A., Happe, L.: ModelJoin. A Textual Domain-Specific Language for the Combination of Heterogeneous Models. Tech. Rep. 1, Karlsruhe Institute of Technology, Faculty of Informatics (2014). URL `http://digbib.ubka.uni-karlsruhe.de/volltexte/1000037908`
17. Burger, E., Henß, J., Küster, M., Kruse, S., Happe, L.: View-Based Model-Driven Software Development with ModelJoin. Software & Systems Modeling **15**(2), 472–496 (2014)
18. Burger, E., Schneider, O.: Translatability and Translation of Updated Views in ModelJoin. In: P. van Gorp, G. Engels (eds.) Proceedings of the 9th International Conference on the Theory and Practice of Model Transformations (ICMT), pp. 55–69. Springer (2016)
19. Chechik, M., Nejati, S., Sabetzadeh, M.: A Relationship-based Approach to Model Integration. Innovations in Systems and Software Engineering **8**(1), 3–18 (2012)
20. Cicchetti, A., Ciccozzi, F., Leveque, T.: A Hybrid Approach for Multi-view Modeling. ECEASST **50** (2011)
21. Combemale, B., DeAntoni, J., Baudry, B., France, R.B., Jézéquel, J., Gray, J.: Globalizing Modeling Languages. IEEE Computer **47**(6), 68–71 (2014)
22. Czarnecki, K., Helsen, S.: Feature-based survey of model transformation approaches. IBM Systems Journal **45**(3), 621–646 (2006)
23. Debreceni, C., Horváth, A., Hegedüs, A., Ujhelyi, Z., Ráth, I., Varró, D.: Query-driven Incremental Synchronization of View Models. In: Proceedings of the 2nd Workshop on View-Based, Aspect-Oriented and Orthographic Software Modelling (VAO), pp. 31:31–31:38. ACM (2014)
24. Derler, P., Lee, E.A., Vincentelli, A.S.: Modeling Cyber–Physical Systems. Proceedings of the IEEE **100**(1), 13–28 (2012)
25. Eclipse EMF Facet project. URL `http://www.eclipse.org/facet/`
26. Engels, G., Heckel, R., Taentzer, G., Ehrig, H.: A Combined Reference Model- and View-Based Approach to System Specification. International Journal of Software Engineering and Knowledge Engineering **7**(4), 457–477 (1997)
27. Etzlstorfer, J., Kusel, A., Kapsammer, E., Langer, P., Retschitzegger, W., Schoenboeck, J., Schwinger, W., Wimmer, M.: A Survey on Incremental Model Transformation Approaches. In: Proceedings of the Workshop on Models and Evolution (ME) co-located with MoDELS, pp. 4–13 (2013)
28. Feldmann, S., Wimmer, M., Kernschmidt, K., Vogel-Heuser, B.: A Comprehensive Approach for Managing Inter-model Inconsistencies in Automated Production Systems Engineering. In: Proceedings of the IEEE International Conference on Automation Science and Engineering (CASE), pp. 1120–1127 (2016)
29. Finkelstein, A., Kramer, J., Nuseibeh, B., Finkelstein, L., Goedicke, M.: Viewpoints: A Framework for Integrating Multiple Perspectives in System Development. International Journal of Software Engineering and Knowledge Engineering **2**(1), 31–57 (1992)
30. Goldschmidt, T., Becker, S., Burger, E.: Towards a Tool-Oriented Taxonomy of View-Based Modelling. In: Proceedings of the Modellierung 2012. GI (2012)
31. Golra, F.R., Beugnard, A., Dagnat, F., Guérin, S., Guychard, C.: Addressing Modularity for Heterogeneous Multi-model Systems using Model Federation. In: Companion Proceedings of the 15th International Conference on Modularity, pp. 206–211. ACM (2016)
32. Hidaka, S., Tisi, M., Cabot, J., Hu, Z.: Feature-based Classification of Bidirectional Transformation Approaches. Software and System Modeling **15**(3), 907–928 (2016)
33. Hildebrandt, S., Lambers, L., Holger, G., Rieke, J., Greenyer, J., Schäfer, W., Lauder, M., Anjorin, A., Schürr, A.: A Survey of Triple Graph Grammar Tools. In: Proceedings of the 2nd International Workshop on Bidirectional Transformations (BX), pp. 1–18. EC-EASST (2013)
34. Hilliard, R.: On Metamodels in 42010 (2011). URL `http://www.iso-architecture.org/ieee-1471/docs/Hilliard-On-Metamodels-in-42010.pdf`
35. Hudak, P.: Building Domain-specific Embedded Languages. ACM Comput. Surv. **28**(4) (1996)

36. Jakob, J., Königs, A., Schürr, A.: Non-materialized Model View Specification with Triple Graph Grammars. In: Proceedings of the 3rd International Conference on Graph Transformations (ICGT), pp. 321–335. Springer (2006)
37. Jakob, J., Schürr, A.: View Creation of Meta Models by Using Modified Triple Graph Grammars. Electr. Notes Theor. Comput. Sci. **211**, 181–190 (2008)
38. Jalali, A., Ghamarian, M.A.H., Rensink, D.A.: Incremental pattern matching for regular expressions. In: A. Fish, L. Lambers (eds.) Proceedings of the 11th International Workshop on Graph Transformation and Visual Modeling Techniques (GTVMT), pp. 736:1–736:12. EC-EASST (2012)
39. Jouault, F., Tisi, M.: Towards Incremental Execution of ATL Transformations. In: Proceedings of the 3rd International Conference on the Theory and Practice of Model Transformations (ICMT), pp. 123–137. Springer (2010)
40. Kitchenham, B., Charters, S.: Guidelines for performing Systematic Literature Reviews in Software Engineering. Tech. Rep. EBSE 2007-001, Keele University and Durham University Joint Report (2007)
41. Kolovos, D.S., Paige, R.F., Polack, F.A.: Merging Models with the Epsilon Merging Language (EML). In: Proceedings of the 9th International Conference on Model Driven Engineering Languages and Systems (MoDELS), pp. 215–229. Springer (2006)
42. Kolovos, D.S., Rose, L.M., Matragkas, N.D., Paige, R.F., Polack, F.A., Fernandes, K.J.: Constructing and Navigating Non-invasive Model Decorations. In: Proceedings of the 3rd International Conference on Theory and Practice of Model Transformations (ICMT), pp. 138–152. Springer (2010)
43. Krahn, H., Rumpe, B., Völkel, S.: MontiCore: a Framework for Compositional Development of Domain Specific Languages. International Journal on Software Tools for Technology Transfer **12**(5), 353–372 (2010)
44. Langer, P., Wieland, K., Wimmer, M., Cabot, J.: EMF Profiles: A Lightweight Extension Approach for EMF Models. Journal of Object Technology **11**(1), 1–29 (2012)
45. Langlois, B., Exertier, D., Zendagui, B.: Development of Modelling Frameworks and Viewpoints with Kitalpha. In: Proceedings of the 14th Workshop on Domain-Specific Modeling (DSM), pp. 19–22. ACM (2014)
46. Leblebici, E., Anjorin, A., Schürr, A.: Developing eMoflon with eMoflon. In: Proceedings of the 7th International Conference on the Theory and Practice of Model Transformations (ICMT), pp. 138–145. Springer (2014)
47. Linington, P.F.: RM-ODP: the architecture. In: Open Distributed Processing, pp. 15–33. Springer (1995)
48. Linnington, P., Milosevic, Z., Tanaka, A., Vallecillo, A.: Building Enterprise Systems with ODP - An Introduction to Open Distributed Processing. CRC Press (2011)
49. Maoz, S., Ringert, J.O., Rumpe, B.: Synthesis of Component and Connector Models from Crosscutting Structural Views. In: Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE), pp. 444–454. ACM (2013)
50. Martinez, S., Garcia-Alfaro, J., Cuppens, F., Cuppens-Boulahia, N., Cabot, J.: Model-driven Extraction and Analysis of Network Security Policies. In: Proceedings of the 16th International Conference on Model Driven Engineering Languages and Systems (MoDELS), pp. 52–68. Springer (2013)
51. Mayol, E., Teniente, E.: A Survey of Current Methods for Integrity Constraint Maintenance and View Updating. In: Proceedings of Advances in Conceptual Modeling (ER'99): Workshops on Evolution and Change in Data Management, Reverse Engineering in Information Systems, and the World Wide Web and Conceptual Modeling, pp. 62–73. Springer (1999)
52. OMG Model Driven Architecture. URL `http://www.omg.org/mda/`
53. Melnik, S., Bernstein, P.A., Halevy, A., Rahm, E.: Supporting Executable Mappings in Model Management. In: Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD), pp. 167–178. ACM (2005)
54. Mens, T., Van Gorp, P.: A taxonomy of model transformation. Electronic Notes in Theoretical Computer Science **152**, 125–142 (2006)
55. Moreno, N., Romero, J.R., Vallecillo, A.: An Overview Of Model-Driven Web Engineering and the MDA. In: Web Engineering: Modelling and Implementing Web Applications, Human-Computer Interaction Series, pp. 353–382. Springer (2008)

56. Nassar, M.: VUML: a viewpoint oriented UML extension. In: Proceedings of the 18th IEEE International Conference on Automated Software Engineering (ASE), pp. 373–376 (2003)
57. Noyrit, F., Gérard, S., Selic, B.: FacadeMetamodel: Masking UML. In: Proceedings of the 15th International Conference on Model Driven Engineering Languages and Systems (MoDELS), pp. 20–35. Springer (2012)
58. Palpanas, T., Sidle, R., Cochrane, R., Pirahesh, H.: Incremental Maintenance for Non-Distributive Aggregate Functions. In: Proceedings of 28th International Conference on Very Large Data Bases (VLDB), pp. 802–813. Morgan Kaufmann (2002)
59. Rubin, J., Chechik, M., Easterbrook, S.M.: Declarative Approach for Model Composition. In: Proceedings of the 2008 International Workshop on Models in Software Engineering (MiSE), pp. 7–14. ACM (2008)
60. Sabetzadeh, M., Easterbrook, S.: View Merging in the Presence of Incompleteness and Inconsistency. Requirements Engineering **11**(3), 174–193 (2006)
61. Schürr, A.: Specification of Graph Translators with Triple Graph Grammars. In: Proceedings of the 20th International Workshop on Graph-Theoretic Concepts in Computer Science (WG), pp. 151–163. Springer (1994)
62. Semeráth, O., Debreceni, C., Horváth, Á., Varró, D.: Incremental backward change propagation of view models by logic solvers. In: Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems, pp. 306–316. ACM (2016)
63. Eclipse Sirius project. URL `https://eclipse.org/sirius/`
64. Troya, J., Brunelière, H., Fleck, M., Wimmer, M., Orue-Echevarria, L., Gorroñogoitia, J.: ARTIST: Model-Based Stairway to the Cloud. In: Proceedings of the Projects Showcase, part of the Software Technologies: Applications and Foundations 2015 federation of conferences (STAF), pp. 1–8 (2015)
65. Ujhelyi, Z., Bergmann, G., Hegedüs, Á., Horváth, Á., Izsó, B., Ráth, I., Szatmári, Z., Varró, D.: EMF-IncQuery: An Integrated Development Environment for Live Model Queries. Science of Computer Programming **98**, 80–99 (2015)
66. Vallecillo, A.: On the Combination of Domain Specific Modeling Languages. In: Proceedings of the 6th European Conference on Modelling Foundations and Applications (ECMFA), pp. 305–320. Springer (2010)
67. Vangheluwe, H., Amaral, V., Giese, H., Broenink, J., Schätz, B., Norta, A., Carreira, P., Lukovic, I., Mayerhofer, T., Wimmer, M., Vallecillo, A.: MPM4CPS: Multi-Paradigm Modelling for Cyber-Physical Systems. In: Joint Proceedings of the Doctoral Symposium and Projects Showcase Held as Part of STAF 2016 co-located with Software Technologies: Applications and Foundations (STAF 2016), pp. 40–47 (2016)
68. Wohlin, C.: Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering (EASE), pp. 38:1–38:10. ACM (2014)
69. Wood-Harper, A., Antill, L., Avison, D.: Information systems definition: the multiview approach. Computer science texts. Blackwell Scientific (1985)
70. Zachman, J.A.: A Framework for Information Systems Architecture. IBM Systems Journal **26**(3), 276–292 (1987)
71. Zito, A., Diskin, Z., Dingel, J.: Package Merge in UML 2: Practice vs. Theory? In: Proceedings of the 9th International Conference on Model Driven Engineering Languages and Systems (MoDELS), pp. 185–199. Springer (2006)