A Characterization of Integrated Multi-View Modeling in the Context of Embedded and Cyber-Physical Systems

Magnus Persson, Martin Törngren, Ahsan Qamar, Jonas Westman and Matthias Biehl
KTH, Stockholm, Sweden
magnper@kth.se, martin@md.kth.se, ahsanq@kth.se, jowestm@kth.se, biehl@md.kth.se

Stavros Tripakis
University of California, Berkeley, USA
stavros@eecs.berkeley.edu

Hans Vangheluwe
University of Antwerp, Belgium and McGill University, Montreal, Canada
hv@cs.mcgill.ca

Joachim Denil
University of Antwerp, Belgium and McGill University, Montreal, Canada
joachim.denil@cs.mcgill.ca

ABSTRACT
Embedded systems, with their tight technology integration, and multiple requirements and stakeholders, are characterized by tightly interrelated processes, information and tools. Embedded systems will as a consequence be described by multiple, heterogeneous and interrelated descriptions such as for example requirements documents, design and analysis models, software and hardware descriptions. We refer to a system designed this way as a multi-view (MV) system.

The main contribution of this paper is a characterization of model-based approaches to MV systems. The characterization takes three main perspectives for the relations between viewpoints: semantic relations (content), relations over time (process), and manipulation of views (operations). We complement these perspectives by investigating MV system challenges and by a survey of related approaches. The characterization aims to provide a basis for a better understanding, design and implementation of MV systems, and thereby to overcome the current fragmented points of view on integrated multi-view modeling (MVM).

Keywords
views, viewpoints, consistency, multi-view modeling (MVM), survey, embedded systems, view relations, viewpoint relationship patterns

1. INTRODUCTION
Embedded systems are characterized by a tight integration of multiple technologies and have to meet many requirements, for example on robustness, dependability and maintainability. Embedded systems development involves many stakeholders and tightly interrelated processes, information and tools, and is therefore setting a for concurrent engineering. The growing applicability of embedded systems, due to their versatility, increased connectivity and improved price-performance, leads to a further increase of stakeholders and concerns, as illustrated by the emerging areas of cyber-physical systems and the Internet of things.

Embedded systems are as a consequence specified through multiple and interrelated descriptions such as requirements documents, design and analysis models, software/hardware descriptions and product configurations. We call such descriptions views and use the terminology from ISO42010 [4]. A view is a representation of a system from the perspective of a related set of concerns. The concept of concerns is rather open in the ISO42010 definition. Since a concern relates to a description which is of relevance to a stakeholder, we consider the following:

- A concern will be related to a set of system parts (a subset of functionalities and/or components of the system).
- A concern will be related to one or more aspects of the system (properties and qualities of a system e.g. in terms of performance and robustness).
- A concern will be related to the tasks of the stakeholder (e.g. the description that might be used to support design or testing).

A viewpoint is a specification of the conventions for using a view, by establishing the purposes and audience for a view, and the techniques for its creation and analysis. This for example implies that the choices of modeling languages for a view should be driven by the context of the design tasks at hand. While the ISO standard acknowledges the need to deal with relations between views, it does not give any concrete guidance on how to go about addressing this problem.

Any complex system will involve multiple stakeholders and thus multiple views. As a consequence, multi-view (MV) systems have been studied in many different areas (e.g. in requirements, systems and software engineering, enterprise information systems, and in model-driven engineering). For example, the so called Controlled Requirements Expression (CORE) from the early 80s highlights the concept of viewpoints as part of its method for determining functional re-

1 A successor of the IEEE 1471 guideline
requirements. CORE uses traditional structured analysis and design techniques in terms of data- and control-flow diagrams to identify relations between views. [48]. A commonly studied challenge is managing consistency between multiple views, [28, 30, 13, 25, 31]

As embedded systems generally are described from more perspectives than other computer systems, we consider the challenges of multi-view integration to be especially important in this area. Embedded systems development environments are in general significantly heterogeneous, with multiple formalisms and viewpoints that describe systems at different levels of abstraction. Finally, embedded systems, due to their interaction with the physical world, are hybrid systems with multiple aspects of behavior. All these aspects make multi-view modeling especially challenging for embedded and cyber-physical systems.

1.1 Contribution

In our work with embedded systems we have identified the need for establishing a common ground that provides a basis for better understanding, design and implementation of MVM. We focus on a model-based setting, i.e. multi-view modeling (MVM), where views are represented by models. We believe that a systematic approach will be increasingly important in the evolution of embedded systems into cyber-physical systems. The main contribution of this paper is a characterization of MV systems in terms of the relations between views:

- **Content** – Syntactic and semantic relations.
- **Process** – Relations over time and causality.
- **Operations** – To view the MV system in terms of operations.

We complement the characterization by providing a survey of approaches to integrated MVM in section 6. We also discuss challenges in using MV systems (section 3). A brief description of related assessments of MVM is provided (section 1.2). To set the scene for our work, we provide an example of a MV system which is used throughout the paper for illustrative purposes (section 2) and to introduce the basic terminology (section 4) and discussion on the types of relations between view (section 5). We end the paper with a discussion (section 7) and conclusions (section 8).

This paper does not constitute a detailed survey. We strive to give an overview understanding, and do not go into specific details but rather discuss approaches to multi-view modeling as a whole. It is also not a purpose to describe how semantics of views and relations between them are best to be represented.

1.2 Related Work

To the best of our knowledge, few other publications explicitly focus on comparing and classifying different approaches to MVM. Most deal with a specific approach only, but some do so combined with a classification of related approaches. Reiter et al [52] classify transformations into six groups: check, rewrite, translation, fusion, relation, and generation. Boronat et al [16] distinguish between three main approaches: a system model approach where all individual viewpoints are translated into a common system model, a model-driven architecture approach where model transformations handle consistency on a (typically) syntactic level, and a heterogeneous semantics and development approach where each viewpoint is given a mathematical semantics, and the consistency is analyzed. [35] provides a classification of different MV integration challenges, and list three fundamental techniques employed to approach integration: decomposition/composition, abstraction/enrichment, and parallel-development/consistency. Further related work is provided through the survey in section 6. Terminology and formalization of concepts vary widely among the different approaches. In summary, while many useful proposals for MVM approaches have been made, they provide fragmented perspectives to MVM.

2. EXAMPLE

We illustrate the challenges of building MV systems with an example based on an intelligent windshield wiper given in Figure 1. The choice of viewpoints and views in the example is intentionally incomplete; it serves an illustrative purpose rather than being complete. A number of views of the system are available with color-marked cross-dependencies: naming of hardware nodes of the system in Figures 1(a), 1(b) and 1(d) where slightly different naming variants are used, the coupling between the ports of the software units, the mode transitions (sensor values and buttons trigger mode changes) and the output signal sent to the Motor Driver, all in Figures 1(c) and 1(e).

Figure 1: Some views describing the example windshield wiper system.
relationships will appear: for example, a control and timing viewpoint have an implicit relation as further elaborated in [24], since control stability is highly dependent on timing. On the other hand, timing is also related to other views, such as the actual implementation of the code, through its execution speed. Such a relation is typically not intuitively clear to the developer(s). Extending this discussion, in the example there are other more subtle relations that depend on other things than content. For example, the software views and hardware views can be expected to be built before the allocation view, so the allocation view is built based on information already available in the former.

Finally, it is quite likely that some views are not built manually, but rather created in an automated fashion, for example through model transformation. In the above example, the hardware view can initially be built as a copy of the network diagram.

3. CHALLENGES OF MV MODELING

Multi-view systems will inherit the complexity of the embedded systems they are intended to describe. When splitting the system description into multiple views, the relations between the different views need to be dealt with. For non-trivial systems it will no longer be sufficient to resolve the gaps between views over a cup of coffee. Perry and Wolf discuss this problem of views that drift apart as architectural drift, which may lead to a deterioration and model erosion [49, 58]. Hence, there are a number of challenges in MVM.

We have identified the following group of main concerns and challenges for MV systems:

- **Consistency.** There is a need to ensure that the different views are consistent with each other, i.e. that they do not contradict each other.
- **Traceability.** A common need is the ability to perform tracing between views, e.g. to trace between corresponding elements in different views.
- **Reuse.** To reuse of content from one view should be as easy as possible in another, new view. This can be done in different manners – ideally by referencing the other view, less ideally by copying and pasting into a new model, such that updates can not be propagated.
- **Automation.** MV systems will be built in a certain tool environment. There is need for explicit support to automate or otherwise facilitate a development process involving multiple stakeholders and views, and coordinating their evolution.
- **Change propagation.** Decisions and changes in one view may either be of local nature or may affect one or more other views, creating the need for manual or automated change propagation. A slightly simpler variant is support for impact analysis, i.e. only finding where updates are needed.
- **Extendability.** Openness to accommodating changes to the representation of the MV system, such as the inclusion of a new view or change of a viewpoint.

Dealing with MVM is complex, with (i) multiple types of contextual settings in terms of processes and legacy, (ii) multiple heterogeneous viewpoints and views, (iii) multiple types of relations among views, and (iv) multiple ways of dealing with such relations. Supporting them in a multi-tool environment will require addressing all classical tool integration aspects [61, 8, 17]. Derived challenges for MVM include that of designing such systems, dealing with legacy systems, and in supporting automation for such systems, while ensuring that the above mentioned challenges are properly handled. Unfortunately, our experience is that tool and tool integration support today is ad-hoc, and there is a lack of established standards. Within specific domains, such as requirements engineering, and embedded software design, such tools are well integrated.

4. VIEWS AND VIEWPOINTS

We here more formally define main characteristics of views and viewpoints. Views are intended as abstractions of a given system $s$. Since we in this paper only deal with model-based approaches, we can make the assumption that a view $V$ further is represented by a model $M$. A model $M$, or a group of models $M_1, M_2, \ldots$, has certain properties.

- **Faithfulness.** A view $V$ is defined by its model $M$, and the view’s mapping of the model to a certain semantics $S(V) = \{s_1, s_2, \ldots\}$, i.e. a set of systems. The set of systems can be specified either explicitly or through constraints on the system in the model $M$. We only consider the view to be faithful representation of the intended system $s$ iff $s \in S(V)$.
- **Consistency.** Given a set of internally consistent views $V_1, V_2, \ldots, V_n$, does there exist any system $s$ such that each $V_i$ is a faithful view of $s$? If no such $s$ exists, then the views are inconsistent.
- **Completeness.** Given an intended level of abstraction of a system $s$, defining each view independently is sometimes sufficient in order to completely define the system ($\bigoplus V_i s = s$, where $\oplus$ denotes composition). Such a system has a complete set of views.

We define a viewpoint $V$ as the mapping of a meta-model (or other grammar defining the modeling language syntax) $M$ to a semantic domain $S(V)$, where $V \in V, M \in M$ and $S(V) \subseteq S(M)$ and $S(V) \subseteq S(M)$. The semantics of a specific viewpoint can be seen as a subset of the language semantics, as was also noted in [34].

The point of view presented above is significantly different from traditional language engineering, where the basic assumption is that the semantics are coupled to the language itself. However, this clearly does not hold in a MV setting. Consider a simple box-and-lines diagram: it will have a vastly different meaning depending if it is titled “Hardware structure” or “Software structure”.

In other words, we see a meta-model as the set of all possible model instances it can be instantiated to, and a viewpoint as the set of all possible view instances. A semantic domain is then the set of semantics that is possible to express in the viewpoint. Hence, these concepts are coupled as illustrated in Figure 2. Of course, given that viewpoints are often defined informally, these semantic mappings are often

---

2A model can of course be thought of as a group of sub-models, to more closely mirror ISO 42010 where a view may consist of multiple models.
only implicit.

We relate back to the example in section 2 in order to clarify the concepts. Each view is there given with a model. For the network diagram, a viewpoint with the semantic domain of all possible network configurations is used. The semantics is given by all systems with the network topology as given by the diagram. For the mode view, the semantic domain is all arbitrary sequences of modes, while the semantics is the valid subset as specified by the automaton. For example the sequence “Off” → “Auto/On” is not possible, neither is transition to the mode “Illegal” since it is not included.

5. VIEW RELATIONSHIPS

Views of the same system, and viewpoints in general, are normally not entirely orthogonal, but have relations to each other. These relations are caused by overlap in the concerns that guide the viewpoint definitions, data which is shared over several views and through process constraints.

In this paper, we classify different view relations and viewpoint relation patterns. View relations are relations that exist between views. In some cases, such relations can be generalized and applied to all view instances of a pair of viewpoints. We call this a relationship pattern, in order to clearly distinguish between the two concepts. We will hence, for example, talk about “projections” between concrete views and “projection patterns” between the viewpoints that guide these views. Formally: if \( V_1 \in V_1 \) and \( V_2 \in V_2 \) are views of the same system \( s \), \( V_1 \sqsupset V_2 \Rightarrow V_1 \sqsupset V_2 \). The symbol \( \sqsupset \) denotes the relation/relationship pattern under consideration. We categorize the types of relations/relationship patterns into three main groups: content, process, and operations.

5.1 Content

The first group of relationships and relationship patterns that we describe relate to the content of the views, as illustrated in Figure 3.

The overlap of the content may be either purely semantic, or both semantic and syntactic. If the overlap is syntactic, that means that the two views use the same modeling language (or parts thereof). This also implies that a purely syntactic overlap is not that interesting. That two views use the same modeling language is not relevant if they don’t share the wording, “relationship pattern”, is inspired by the wording used in dependency modeling[31].

meaning too, i.e. that there is a semantic overlap (compare figure 3). There can be semantic relations without any syntactic relation existing, e.g. if the two viewpoints are based on formalisms with completely separate syntax. With this we describe a number of view relations:

- **Orthogonal/Independent** means that it is known that the views \( V_1, V_2 \) have no direct semantic relationship, i.e. \( S(V_1) \cap S(V_2) = \emptyset \). We also write \( V_1 \perp V_2 \). Indirect relationships, going through a third view, may be possible but no direct ones.

- **Syntactic overlap** means that two views \( V_1, V_2 \) have some information that is overlapping syntactically, but may also have some information that is unique to each view that is not sharing semantics. This conceptually means that \( S(V_1 \setminus V_2) \cup S(V_1 \cap V_2) \cup S(V_2 \setminus V_1) \), and that a syntactic overlap \( M_1 \cap M_2 \) can be identified.

- **Semantic overlap** is used to classify where it is known (or possibly only assumed) that there is a semantic relation between two views \( V_1, V_2 \), i.e. \( S(V_1) \cap S(V_2) \neq \emptyset \), but where there is no syntactic overlap. There may still be an explicit way to handle the relation, e.g. through a suitable operation or by explicitly modeling the semantic mapping for both involved viewpoints.

We distinguish one special case: Two views are semantically equivalent if they have the same semantics, i.e. \( S(V_1) = S(V_2) \).
Another type of process-related relations refers to process-related relations as follows: A view \( V_1 \) is an abstraction of another view \( V_2 \) if its semantics is a subset of the other, i.e. \( S(V_1) \subseteq S(V_2) \). An abstract model represents a broader group of systems, since it contains less information. We may also say that \( V_2 \) is a refinement of \( V_1 \) or write \( V_1 \models V_2 \). For a detailed discussion of different variants of abstraction see [54]. Refinement may both be homogeneous (using the same formalism, providing a more detailed description) or heterogeneous.

We distinguish one case especially: filter/subset, as a relation where the more abstract view \( V_1 \) is a strict syntactical subset of the refined view \( V_2 \), i.e. \( M_1 \subset M_2 \) and \( S(M_1) \perp S(M_2 \setminus M_1) \). An example may be to filter a specific concern, e.g. to single out mechanical aspects from a model covering both mechanic and electric properties. In the opposite direction we speak about extension.

**Association** is a relation where an association view is used to connect two or more base views. Examples include allocation views (between hardware and software views), dependency models between views, or design contract views [24] between e.g. a control and a software implementation view. As such, the association view either bind the views together, or constrain the shared semantics.

### 5.2 Process

Process-related relations are based on how the views relate to the developmental workflow, as illustrated in Figure 4. In the process perspective, there is an important difference between refinement and abstraction: the implied workflow direction differs. For the workflow constraints, we classify process-related relations as follows:

- **Precedence** – the first view is expected to exist before the second. They don’t necessarily share data.
- **Dependency** – the second view contains data that is an output from the first (thus constituting a precedence relation with content overlap).
- **Co-Dependency** – two views each contain data from the other (i.e. a circular dependency). There is a possible need for multiple iterations needed to reach a fix-point.

Another type of process-related relations refers to versions and variants of views. Versions are successive iterations of the same view, while variants are alternatives of the same view (e.g. for product line or configurability reasons). Such variability may be handled internally in the views or externally.

### 5.3 Operations

Another way to study MV systems is to study operations on views, e.g. analysis, consistency checking, view composition, projection, as illustrated in Figure 5. Here we categorize such operations and how they may be implemented.

Operations on views may take one or several inputs. The perspective of viewing MV environments focused on the operations between the different views places the emphasis on the mechanisms, rather than on the content. Some such operations are semantics-preserving, others are not. Some concrete examples of types of operations are given below:

- **Composition**, \( \oplus \), involves defining an operation creating a common view encompassing the semantics of two or more incoming views, i.e. \( V_{A\oplus B} = V_A \oplus V_B \).
  - **Communication** is a special case of composition, where the two views are heterogeneous but communicating with each other. For example a view of a control system and its environment.
  - **Merging** can be seen as variant of composition, where the two views to be merged represent different variants of the same system.

- **Projection** involves to remove part of the syntactical content in a view in order to reach a reduced one.
- **Extension** is the opposite of projection – to add additional syntactical detail to an already existing view.
- **Analysis** is an operation working on one (or a few views) in order to provide a resulting data view, for example true/false, utilization rates, efficiency numbers or a Pareto front in design space exploration.
  - **Diffing** is a special case – a comparison between different variants of the same view.
- **Synthesis** refers to generation of lower-level implementation based on one or several source views.
- **General case**. Of course it is also possible to create custom translation rules between different views. This can be done e.g. through a general-purpose transformation language.

### 5.4 Illustration of the Perspectives

The content, process and operations perspectives on MVM are not entirely exclusive to each other, and it is hard to say that there is a hard line between them. They rather represent different mindset and focus. This is exemplified by the recurring reference to the relation between the hardware view (Figure 1(a)) and network diagram (Figure 1(b)) of the windshield wiper example above. For the content perspective, it is an example of a refinement/abstraction relation.
Once again, in the example: Given an assumed design process which starts by creating the network diagram, there would be a dependency from the network diagram to the hardware view. An analysis approached from the operations perspective says that this operation is an extension: start from network diagram and add missing information. Later in the process, the opposite operation, filtering, could of course also be applied.

6. APPROACHES TO MVM

We provide a survey of state of the art approaches to MVM. Other common terms found in the literature for this kind of development include multi-viewpoint, multi-paradigm, multi-modeling, multi-model, heterogeneous modeling and multi-formalism development. Our survey is coarse-grained. Its purpose is not to provide a detailed comparison between every single variant of each approach, but to capture the main characteristics of typical work in the approach. The approaches are also not mutually exclusive and some areas build upon others. We have surveyed several fields of study from model-driven engineering, over mechatronics and systems engineering, to tool integration.

A comparison of the different approaches is shown in Table 1, based on the challenges in section 3 and the perspectives in section 5. For the challenges, the marking indicates our assessment of how well the approach handles the challenge. Lack of marking implies that the approach neither solves the issue nor makes it impossible to do so (e.g. by combining the approach with one of the others). Only if it goes against the spirit of the approach, or its fundamental implications make it infeasible, we mark a challenge as explicitly excluded using a cross.

For the criteria based on the three perspectives on view relations in MVM, the marking indicates that the approach explicitly handles such relations/relationship patterns between arbitrary views/viewpoints. In some cases, the approach may assume that views have a certain relation. This is the case for e.g. architecture frameworks, which use a pre-defined set of viewpoints. Few relations have been marked here and only as partially supported, since they are only available for the viewpoints in question and not in general.

6.1 MDE, Meta-Modeling and Model Transformations

Model transformation [14] is a mechanism for describing and enforcing relations in MVM. Once the relation between different models is analyzed it can be described in terms of model transformations. Model transformations provide a precise and executable description. They can be used to create new models that fulfill the relation or to enforce a relation between existing models.

6.1.1 Common Shared Meta-Model

A common approach to handle the problem of linking several models is to try to build a common core meta-model encompassing the key content of the different views to be integrated. Such a common meta-model might be used for checking consistency or as a pivot meta-model for model exchange between the views e.g. through model transformations, see e.g. [9, 13]. Standard exchange formats can actually be seen as de-jure or de-facto versions of common meta-models. A strategical question in the design of a shared meta-model is how inclusive it should be. Making the shared meta-model smaller makes it simpler, less complex and easier to understand, but it also implies more information loss.

6.1.2 Mega-modeling

Different definitions and applications of mega-modeling exist in the literature [12, 20]. Hebig et al. unify the different definitions and intentions of mega-modelling with the following definition: a model that contains models and relations between them [39]. From the perspective of MVM, mega-modelling has indeed some benefits because a mega-model is situated at the modelling language level rather than at the model instance level. Mega-models can precisely capture the relations between different models and languages, i.e. views. In mega-modelling all relations between models and languages can be represented by adding different types of relations to the mega-modelling language. A common relation in MVM is for example the $\text{isTransformedTo}$ relation, denoting the transformation from one view to another.

6.2 Multi-Viewed Components

In the area of component-based design, there is some work on multi-viewed component models, focusing on composition and reuse of different components. Related to components are interface theories, which formally describe the interaction between different components. Henzinger [41] defines a group of interface theories, and review the conditions needed to ensure that they are orthogonal. HRC (heterogeneous rich components) is a component-based approach with contracts [22, 11] using hybrid automata as interface formalization. The approach has four predefined viewpoints, and additional ones can be added. However, the different viewpoints are not orthogonal to each other; they all share the same meta-model and semantic domain and the viewpoints are rather an informal categorization. De Lara et al [23, 38] describe a component model where the consistency between views is ensured by gluing them through relating them to each other with triple graph grammars, a type of model transformations. Bureš et al [18] suggest an approach where different types of components form families of component types.

6.3 Aspect-Oriented Modeling

Aspect-oriented modeling (AOM) has its origins in Aspect-Oriented Programming (AOP) and is commonly used for software engineering purposes. OAM allows for separate expression of multiple concerns. These aspects are later again integrated through weaving. Aspects can entail security, timing, or other extra-functional concerns. [32, 42, 20]. Advanced AOM tools are now available, e.g. TouchRAM [5].

A major challenge of the AOM is the point-cut problem, i.e. the challenge of ensuring that the weaving is not affected by changes in the model. A challenge is also ensuring that the different aspects do not influence each other (i.e. normally it is assumed that they are orthogonal).

6.4 Architecture Frameworks

A classical approach is to define a predefined set of views, a so called view framework [46, 4]. A long list of architecture
Approach | MDE | - Shared MM | - Mega-modeling | MV Components | Arch. FWs | System languages | Tool Integration | Category Theory | Ontologies | Dependency Mod. | Model versioning | Co-Simulation
---|---|---|---|---|---|---|---|---|---|---|---|---
Consistency | ○ | ● | ? | ○ | ○ | ○ | ● | ● | ● | ● | ○ | ○
Traceability | ● | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○
Reuse | ● | ○ | ? | ● | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○
Automation | ○ | ○ | × | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○
Change Propagation | ○ | ○ | × | ○ | ○ | ○ | ● | ○ | ○ | ○ | ○ | ○
Extendability | ○ | × | ? | ○ | × | ○ | × | ○ | ○ | ○ | ○ | ○
Orthogonal | ○ | ● | ? | ○ | ○ | ● | ● | ● | ● | ● | ● | ●
Syntactic overlap | ● | ● | ? | ○ | ● | ● | ● | ● | ● | ● | ● | ●
Semantic overlap | ○ | ● | ? | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○
Refinement/Abstr. | ○ | ○ | ? | ○ | ○ | ○ | ○ | ○ | ● | ○ | ○ | ○
Association | ○ | ○ | ? | ○ | ● | ● | ● | ● | ○ | ○ | ○ | ○
Precedence | ○ | ○ | ○ | ● | ● | ● | ● | ● | ● | ● | ● | ●
Dependency | ○ | ○ | ○ | ● | ● | ● | ● | ● | ● | ● | ● | ●
Co-Dependency | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ● | ● | ●
Versions | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ● | ● | ●
Variants | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ○ | ● | ● | ● | ●
Composition | ● | ○ | ○ | ● | ● | ● | ● | ● | ● | ● | ● | ●
Projection | ● | ○ | ○ | ● | ● | ● | ● | ● | ● | ● | ● | ●
Extension | ● | ○ | ○ | ○ | ● | ● | ● | ● | ● | ● | ● | ●
Analysis | ● | ○ | ? | ○ | ● | ● | ● | ● | ● | ● | ● | ●
Synthesis | ● | ○ | ● | ○ | ● | ● | ● | ● | ● | ● | ● | ●
General case | ● | ○ | ● | ○ | ● | ● | ● | ● | ● | ● | ● | ●

Table 1: Comparison of different approaches to MVM.

frameworks is given in [3]. Although using a standardized set of viewpoints restricts development flexibility, it also has advantages: standardization means that the viewpoints used throughout an industry will be consistent, implying their semantics will be better known, and it becomes more easy to rely on informal methods such as synchronization over coffee for ensuring view consistency.

6.5 System Modeling Languages

There are many modeling languages with some support for MVM. The following discussion only covers SysML, but similar discussions apply also to e.g. ACME [13] or AADL [1]. SysML comes with a number of predefined views. The language also supports explicit description of relationships between modeling elements and packages. For example, the requirements modeling within SysML incorporates predefined relations such as satisfy and allocation. SysML can be said to provide a basic framework and possibilities for extensibility of MV systems (such as profiles based on its UML roots). A view may reference languages and methods in another viewpoint allowing explicit mappings between viewpoints. In [55], for example, the use of SysML and graph transformations are used to support MVM.

6.6 Process Modeling and Tool Integration

Process meta-models have been compared in [40] and specifically process models based on UML have been studied in [10]. Traditionally such languages have been used only to describe processes but they are being extended in such a way that they are becoming executable. We focus on approaches within MVM where a representative example is given by the Formalism Transformation Graph and Process Model (FTG+PM) [45]. The FTG+PM combines a mega-model with a process model. The mega-model captures the involved languages and transformation definitions. The process model, modelled with UML activities, uses instances of the languages and transformations in the mega-model to create a process with control- and dataflow. FTG+PM models can be enacted.

The Tool Integration Language (TIL) [15] is a domain-specific modeling language for describing MVM environments consisting of commercial off-the-shelf development tools, which are often unconnected. TIL is used to describe how these separate tools are connected into tool chains. The tool chain can also be analyzed and needed implementation code partly generated, given meta-models and model transformations.

6.7 Category Theory

Category theory is a branch of mathematics with the main emphasis on modeling relations between different types of object groups (called categories), in order to describe the structure of relations between the objects rather than the objects themselves. Category theory is general and abstract. Only few of the works show how to apply the findings practically: An example is [27] which includes models and meta-models for a concrete example. That approach takes an entirely syntactic approach to consistency, as do many other
category-theoretical papers [36, 43]. This means they do not handle relations on a semantic level, such as semantic overlap.

A few examples of category-theoretic approaches with focus on semantics do exist. The foundations of the Rosetta language and semantics [56] are based on a category-theoretic relation between different discrete formalisms in a domain lattice, by using facets which are assumed to be orthogonal. Another example is [16] which defines the concept of a generic multi-modeling language, i.e. a group of coupled modeling languages that have a partially shared semantics. The semantics is given through mappings to the category-theoretic concept of institutions and through semantic connections between different modeling languages. Another approach trying to build a general formal semantics for heterogeneous multi-models is [26].

6.8 Ontologies
An ontology is defined as a “formal, explicit specification of a shared conceptualization” [37]. This shared conceptualization can provide a common reference for several tools [47, 60] and each tool uses a view of the shared conceptualization [33]. Ontologies are usually defined manually, but automated approaches exist as well [57]. Approaches either define a single, comprehensive ontology or a collection of several ontologies. When several ontologies are used, the links between the ontologies need to be defined separately.

Ontology approaches can be combined with model-based approaches. The approach in [53] employs ontologies to reuse model transformations and manage changes in metamodels. In case of large gaps, the data might need to be annotated, as shown in [21].

6.9 Dependency Modeling
Product properties (captured in models) influence each other, referred to as dependencies. The Dependency Modeling Language (DML) [50] is a domain-specific modeling language to support dependency management in MVM environments. The essence of this approach lies in the provision of sufficiently expressive language constructs for modeling dependencies at various levels of detail [51], and dependency patterns, which are patterns that gather and illustrate known dependencies.

Equation-based dependencies or structural dependencies can be captured as dependency patterns, which are executed through model transformations. The dependency models capture relations between models belonging to different viewpoints, and can be used to investigate how such dependencies relate to predictions and decisions. Consistency checking and change management can be supported through dependency models.

6.10 Model Evolution
One of the challenges in model driven development is to handle changes over time, models evolve. In a development process, the change can affect the content, nature, structure or representation of models. Software evolution is a part of the software development life cycle [44] and deals with changes in software. Model evolution is a special case of general software evolution.

We classify model evolution into content related changes, referring to changes of model content such as when a model element is deleted, and metamodel (abstract syntax) related changes. In the latter case it is thus the modeling language that evolves. In both cases, changes can be either local or systemic, see e.g. [6, 19, 7]. Systemic model evolution is handled by means of diffing and merging operations.

6.11 Co-Simulation
Larger systems are typically structured into subsystems to manage their complexity, giving rise to multiple descriptions that will be interacting through horizontal or vertical interfaces. The former type of composition is exemplified by separate models describing an embedded control system and its environment, whereas the latter is exemplified by interactions between abstract behavior models (such as models of application functionality) and the software/hardware platform (represented e.g. by APIs or by abstract properties such as delays) that execute them. Co-simulation provides a very commonly used technique in which the behavior of a composed system can be analyzed. Co-simulation enables early integration, verification and validation of the system behavior and is as such very valuable. Co-simulation does not however explicitly manage other MV relations than composition of behavioral models of directly interacting systems. To support co-simulation, interfaces between models needs to be described. Moreover interaction between simulation engines must be supported, see e.g. [2, 59].

7. DISCUSSION
We have presented a survey of approaches to MVM, based on a qualitative classification using four main groups of criteria: a group of fundamental challenges, and three different perspectives (content, process, operations) that can be applied to the MVM problem.

Prospective implementers and other stakeholders of MVM environments can use the suggested perspectives and the survey in two main ways:

- As a checklist of potential features to include into their environments
- As a design guideline indicating the suitability of different MVM approaches and mechanisms depending on which of the challenges a design environment will have to deal with.

Our comparison of the different approaches shows that they have vastly different focus, both in terms of the attacked challenges and in the way the users are expected to think about the MVM problem. Descriptive approaches such as meta-modeling, modeling languages, dependency modeling and ontologies, provide means to model relations between views, and may also provide the basis for analysis and synthesis based on such models. Certain approaches such as architecture frameworks and system modeling languages go a step further in that they predefine views and/or restrict the formalisms available and hence reduce the problem to handling already known relations. Yet other approaches focus on operations and mechanisms for manipulating views through e.g. model transformations or by supporting their composition.
Strategies for handling view relations in MVM systems are required. Few approaches handle view relations in a well-working manner, as many focus mainly on the content view by view. None do it for all three of the categories that we have listed. To meet all the challenges of relevance for future development environments, a combination of several of the different approaches will be required. Designers of MVM environments should consider the different approaches’ strengths and weaknesses in relation to the requirements they need to meet.

We have further noted that there are several possible alternatives on how to handle consistency, which is one of the main challenges listed: (i) ignoring the problem, (ii) manual checks, (iii) designing the system models in a way such that consistency is ensured, e.g. by relying on an agreed system structure, (iv) linking entities in the different models that are representing the same entity, (v) relying on a single source/master view, and (vi) automatic consistency checking.

A recurring pattern is that several of the challenges can be tackled to different extents. For example, in dependency modeling, the dependency can be modeled with more or less details [51]. In a similar manner, different approaches support tool integration and automation to various extents and levels of syntactic/semantic consistency. There are likely to be trade-offs against for example modeling effort involved in choosing appropriate solutions.

Some issues are present in essentially all of the MVM approaches. Problems such as integrating with legacy tools, versioning, variants and life-cycle issues in general are not really specific to the problem of MVM. However, all these issues become harder with MVM since MV systems imply the need for a large set of tools, formalisms and languages. The same applies to the problem of handling versioning, variability, and concurrent editing on the different models, which are issues that few of the approaches handle.

8. CONCLUSIONS AND FUTURE WORK

Dealing with multi-view modeling will be essential for developing embedded systems of the future. In our work we have many times come across the problem – it is recurring. The apparent lack of an existing taxonomy for relations among views, led us to initiate this effort. The result of the effort is a characterization of MVM, in terms of three perspectives with the intention to improve the understanding of MVM. The scope of this paper has been on model-based approaches to MVM. It is likely that our characterization also is applicable to other approaches.

Avenues for future work include further validation of our characterization, e.g. by performing a complementary survey on the level of individual tools/environments instead of classifying entire approaches. Efficient handling of view relations in current and future tool environments for MVM is a major challenge for the embedded systems community.

Acknowledgement

Part of this work was funded by the European Union ARTEMIS Project MBAT and the Swedish Vinnova project ESPRESSO.

9. REFERENCES
