Supporting the Model-Driven Organization Vision through Deep, Orthographic Modeling

Christian Tunjic*,[a], Colin Atkinson[a], Dirk Draheim[b]

[a] Software Engineering Group • University of Mannheim • Mannheim, Germany
[b] Large-Scale Systems Group • Tallinn University of Technology • Tallinn, Estonia

Abstract. *In a model-driven organization, all stakeholders are able to deal with information about an organization in the way that best supports their goals and tasks. In other words, they are able to select models of the organization at the optimal level of abstraction (e. g. platform independent) in the optimal form (e. g. graph-based) and with the optimal scope (e. g. a single component). However, no approach exists today that seamlessly supports this capability over the entire life-cycle of organizations and the IT systems that drive them. Enterprise architecture modeling approaches focus on supporting model-based views of the static "architecture" of organizations (i. e. enterprises) but generally provide little if any support for operational views. On the other hand, business intelligence approaches focus on providing operational views of organizations and usually do not accommodate static architectural views. In order to fully support the model-driven organization (MDO) vision, therefore, these two "worlds" need to be unified and a common, natural and uniform approach for defining and supporting all forms of views on organizations, at all stages of their life-cycles, needs to be defined and implemented in an efficient and scalable way. This paper presents a vision for achieving this goal based on the notions of deep and orthographic modeling. After explaining the background to the problem and introducing these two paradigms, the paper presents a novel approach for unifying them, along with a prototype implementation and example.*

Keywords. Orthographic System Modeling • Enterprise Architecture Modeling • Business Intelligence

Communicated by T. Clark. Received 2016-10-25. Accepted after 1 revision on 2018-01-23.

1 Introduction

The core idea behind the MDO vision is to allow all stakeholders in an organization to fulfill their assignments using representations of (parts of) that organization that best suit their skills and tasks (Clark et al. 2013). This need occurs across all phases of an organization's life-cycle (from analysis and design to operation and maintenance), at all levels of abstraction (from platform-independent to platform-specific) and for all manner of tasks (from planning and development to delivery and usage). Moreover, it must be supported in the face of constant change across all aspects of the organization's structure, behavior and knowledge (Bittmann 2014).

The key requirement implied by this vision is support for *views* or *perspectives* – that is, representations of (parts of) an organization that let stakeholders see and manipulate its properties in the optimal form for their needs. Since such views provide "a simplified mapping for a special purpose" they conform to the widely accepted definition of "model" by Stachowiak (1973). More specifically, they provide or represent a "mapping to the original", the information they provide is a "reduction of the original" and they are created for highly "pragmatic purposes". The idea of describing a complex architecture via a collection of "models" that each provide a

* Corresponding author.
E-mail. tunjic@informatik.uni-mannheim.de

distinct view of the subject is also proposed in the IEEE1471/ISO42010 standard for Systems and Software Engineering – Architecture Description (IEEE Architecture Working Group 2000; ISO/IEC/IEEE 2011).

To support such a vision across an organization's entire life-cycle some views need to portray relatively static "architectural" aspects of the organization (i. e. its enterprise architecture) while others need to portray relatively dynamic "operation-time" aspects of the system (i. e. business intelligence). The discipline in which the idea of using views to capture the static, architectural properties of an organization is most mature in the Enterprise Architecture (EA) modeling discipline, characterized by approaches such as Zachman (Zachman 1987), TOGAF (The Open Group 2009), RM-ODP (ISO/IEC/ITU-T 1997), Archimate (Iacob et al. 2012) and MEMO (Frank 2002). These all define some kind of "viewpoint framework" defining the constellation of views available to stakeholders and the kind of "models" which should be used to portray them. Some, like Archimate, RM-ODP and MEMO, define their own specialized languages (with multiple sub-languages) to portray views, while others are less prescriptive about precisely what kind of language should be used.

At the operational level, the discipline that focuses on providing operational information (runtime and historical) to business stakeholders of enterprises is commonly known as "business intelligence". Modern business intelligence approaches also rely heavily on the notion of views, but primarily in the form of tables (e. g. spreadsheets) or pictorial visualizations rather than as expressions in formal languages (e. g. process modeling languages, programming languages, ontology modeling languages). Second, business intelligence views tend to be organized and identified in a completely different way to EA modeling views. They are typically defined using multi-dimensional data models (e. g. Online Analytical Processing (OLAP)) in so called "data warehouses" which allow information to be aggregated by users on demand. In contrast, the view types available in EA modeling approaches are usually predefined (i. e. before domain modeling begins) and fixed.

At the present time, there is little commonality between the EA modeling approaches used to describe the static, architectural views of organizations and their IT systems (including software specifications and code), and the business intelligence approaches used to provide operational views. Moreover, transitioning from one to the other at the end of the development phase when a system is first deployed and put into operation, is usually a laborious and error prone process which requires many transformations of information (variously called compilation, deployment and configuration steps). Many observers have recognized that this paradigm shift between the development and operation phases of a system's life-cycle introduces significant accidental complexity and causes many problems (De Lara et al. 2014). A new research area called DevOps has emerged in recent years with the aim of simplifying the process of software deployment and blurring the boundaries between development and operations (Davis and Daniels 2015; Lwakatare et al. 2015). At the model level, similar underlying goals are being explored under the label of "models at run-time" (Aßmann et al. 2014).

In order to realize the full vision of the MDO, therefore, view-based paradigms used in the development and operation phases of a system's life-cycle need to be unified and a common, natural and uniform framework for defining and supporting all views of an organization, regardless of their focus, needs to be defined and implemented in an efficient and scalable way. The premise of this paper is that the optimal way to achieve this is through the integration of two alternative, emerging paradigms for modeling – so called "deep modeling" (De Lara et al. 2014) and "orthographic modeling" (Atkinson et al. 2010). The first of these contributes to the MDO vision by providing a natural way to support "models at run-time" and allows operation and instance data to be incorporated seamlessly into a "multi-level" model. The second contributes to the MDO vision by supporting a natural and

scalable strategy for supporting views, and providing a natural metaphor for navigating around them, that accommodates both the architectural (e. g. EA modeling) and operational (e. g. OLAP) interpretation of views.

The goal of this paper is to present this vision, and demonstrate its practicality through a prototype implementation and a small example. The next section starts by presenting the three main established domains and disciplines that form the background to the approach – enterprise architecture management, business intelligence and multi-level modeling. Sect. 3 then describes the MEMO approach to EA modeling which has pioneered the use of the latter to streamline the integration of, and transition between, architectural and operational views of an organization. Sect. 4 presents the final ingredient for the presented approach by explaining the motivation for, and key ideas behind, the orthographic modeling approach. Sect. 5 then presents the contribution of the paper which is a new, general purpose environment for deep orthographic modeling, which synergetically leverages the deep and orthographic modeling approaches. To demonstrate the conceptual feasibility of the approach and show that it at least has the capabilities of existing methods, Sect. 6 then uses the new environment to model a small example. Finally, Sect. 7 and Sect. 8 conclude with a summary and some closing remarks.

2 Background

In this section we set the scene for the rest of the paper by describing the emerging technologies and disciplines which are relevant to the proposed approach. We first provide overviews of the fields of EA management and business intelligence from the perspective of the viewpoint frameworks they use to organize models. After that we provide an overview of multi-level modeling.

2.1 Enterprise Architecture Management

The importance of Enterprise Architecture Management (EAM) is reflected in the wide range of modeling tools that are marketed as EAM tools (Brand 2015; Roth et al. 2014). The goal of these tools is to impose a certain bookkeeping discipline on enterprise architecture management and ensure that information is only manipulated and updated in appropriate ways. This is clearly shown in the collection of critical features contained in the magic quadrant for enterprise architecture tools (Brand 2015). Essentially, EAM tools represent an IT landscape's meta data repository (or metamodel repository in the terminology of Brand (2015)) and through this facilitate an organization's decision support capabilities, presentation capabilities and various other advanced analysis capabilities.

Deploying an EAM tool within a system landscape initiates a trail of IT system documentation, but this trail exists in its own right and is not genuinely integrated in the IT system landscape. Integration with the rest of the landscape is a crucial problem for the current generation of EAM tools. For example, Brand (2015) states that an EAM tool "must integrate with project and portfolio management (PPM), application portfolio management (APM), governance, risk and compliance (GRC), and IT financial management". However, interoperability with, and traceability against, IT development systems is not among the critical capabilities identified. Consequently, the traditional EAM tool market sticks, non-disruptively, to the established categories of IT tools, projects and work organization.

The sub-discipline of EAM which focuses most strongly on supporting the alignment of all ingredients of an enterprise, including IT systems is EA modeling. The systematic modeling of EAs can be traced back to the introduction of the Zachman Framework in 1987 (Zachman 1987), and since this time a large number of alternative EA modeling frameworks and approaches have been developed ranging from proprietary approaches, e. g., SAP PowerDesigner (SAP 2016) and governmental reference architectures, e. g., FEAF (US Federal Government 2013) to open, consortium-managed standards, e. g., TOGAF (The Open Group 2009). The one thing that they all share in common is reliance on some kind of "viewpoint

framework" to define the constellations of models that should be used to represent an enterprise architecture. Apart from that, however, they differ tremendously in their precise goals, scope and level of detail.

2.2 Business Intelligence

The general term used to describe approaches that focus on providing operational views of organizations and their execution history is "business intelligence". In particular, the consolidation and analysis of operational information is typically referred to as data warehousing (Draheim 2012). The multi-dimensional data model of data warehousing with its specific combination of subject-orientation and time-variance (Codd et al. 1993) has become a central pillar in today's business analytics and decision support (Inmon 1992). Data warehousing offers analysts exactly what they need in order to understand the operational performance of an organization – a transformed, de-normalized presentation of the operational data as an easy-to-explore data universe (i. e. a model that invites the data analyst to delve into it and start navigating: dicing, slicing, aggregating, querying, testing hypotheses and so forth).

The data warehousing paradigm represents a step towards the model-driven organization vision from two different and important perspectives. On the one hand, it demonstrates the power of a multi-dimensional conceptual model for selecting views, and on the other hand it helps to reduce barriers and tensions in the business/IT alignment of today's organizations. However, data warehousing comes with a lot of baggage – namely, the legacy of current enterprise system landscapes. This is unavoidable because data warehousing approaches address how to integrate a multi-dimensional data model into an existing system landscape. This is ultimately the goal of the non-volatile aspect of data warehousing, i. e., ETL (extraction-transformation-loading) (Vassiliadis 2009), data marts, data integration strategies and so forth. However, the pragmatic, engineering flavour of data warehouses that made them such a huge success in the past hinders the transition to the next higher conceptual level, i. e. the level of the model-driven organization.

One of the most important application areas for the data warehousing paradigm is management accounting. Initially, when budgets are first elaborated in the individual departments, accounting information is usually represented in spreadsheets in a multi-dimensional manner. It is then put into the process-oriented ERP (enterprise resource planning) systems (e. g. SAP FI/FM) by hand, because of the ubiquitous ERP/spreadsheet-divide usually favoured in today's organizations (Draheim 2012). The information is then extracted again from the ERP systems, cleansed and transformed into the data warehouse to support business analytic (i. e. the process by which managers make decisions that impact the budgeting process, using for example the spreadsheet cockpits of a rolling budgeting process). The whole process is therefore a big cycle – multi-dimensional budgeting information is captured in spreadsheets, saved in process based ERP systems, extracted into a multi-dimensional data warehouse and finally used for multi-dimensional budgeting work using spreadsheets. The big problem for today's enterprises is that this cycle is slow, error-prone, opaque, complex, unreliable, obfuscated, non-automatic and non-standardized.

2.3 Multi-Level Modeling

One of the main causes of the current complexity in transitioning from the development phase to the operation phase of a system's life-cycle is the "hard" shift in classification levels usually involved (De Lara et al. 2014). This is because today the technologies used to represent information in the two phases are almost always two-level technologies that can only support a single type-level and a single instance-level at a given time. For example, traditional models used in software engineering (e. g. UML diagrams) can usually not be directly instantiated in the tool used to define them (Gerbig 2017). In order to deploy the types described in models they usually need to

be transformed to the types supported in the run-time execution environments of a programming language.

As pointed out by Frank (2014) and others, the key to moving towards an MDO vision where the transition between development (views) and operational (views) is seamless and smooth, is to adopt information-representation technologies which do not require hard shifts between classification levels (Clark et al. 2013; Frank 2016). Such technologies are increasingly being called multi-level or deep modeling approaches in the literature (Igamberdiev et al. 2016; Neumayr et al. 2016). The key characteristic of multi-level modeling technologies is that they can seamlessly support an unlimited number of classification hierarchies without any kind of deployment or compilation step to make types modeled at one classification level available for instantiation at the next classification level. On the contrary, since all levels are "soft", types are instantly available for use at the level below. In recent years a number of multi-level modeling approaches have been published and there is a growing number of multi-level tool implementations (Igamberdiev et al. 2016).

One particular form of multi-level modeling is the so called "deep modeling" approach which supports multi-level modeling using a particular set of inter-related concepts. The first is the Orthogonal Classification Architecture (OCA) (Atkinson and Kühne 2001, 2002) which separates linguistic classification from ontological classification (Kühne 2006) and organizes them in two orthogonal dimensions as shown in Fig. 1. An OCA environment usually has three linguistic levels (L_2–L_0), where L_2 contains the linguistic (meta-)model (i. e. the basic set of concepts which are used to represent the deep model), L_1 contains the domain content (i. e. the deep model containing the user data) and L_0 containing the "real world" objects that are described in the deep model.

The second is the `Clabject` concept which plays the roles of both *Classes* and *Objects* simultaneously. The `Clabject` concept has two sub-classes – `Entity` and `Connection`, which can be used to model entities (cf. classes/objects) and

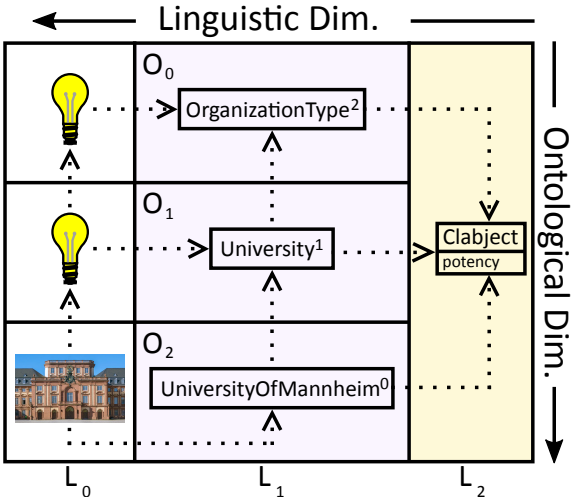

Figure 1: Orthogonal Classification Architecture

connections (cf. associations/links) respectively. Users normally only work with the L_1 linguistic level since this contains the domain content. In Fig. 1 there are three ontological levels (O_0–O_2), but the number of levels is unlimited and can be changed according to the needs of the domain to be modeled. The O_0 level is the most abstract while the O_2 is the least abstract. Generally the O_n level contains the instances of the O_{n-1} level. The user data (i. e. the L_1) is modeled in a unified way using the basic set of model elements defined in L_2.

The third concept is the deep instantiation mechanism, which gives the approach its name. This controls the instantiation of `Clabjects` in the ontological levels within L_1 using a non-negative `Integer` property called `potency`. A `Clabject`'s `potency` governs the extent of its influence over `Clabjects` instantiated from it. An instance of a `Clabject` in the lower (i. e. less abstract) ontological level has a `potency` that is one less than that of the `Clabject`. Since `potency` cannot have a negative value, a `Clabject` with `potency` "0" cannot have further instances in subsequent ontological levels. In the presented example in Fig. 1, the `Clabject` *OrganizationType* in the ontological level O_0 has a `potency` value of "2". This means the model element *OrganizationType* can have instances at the next two ontological levels,

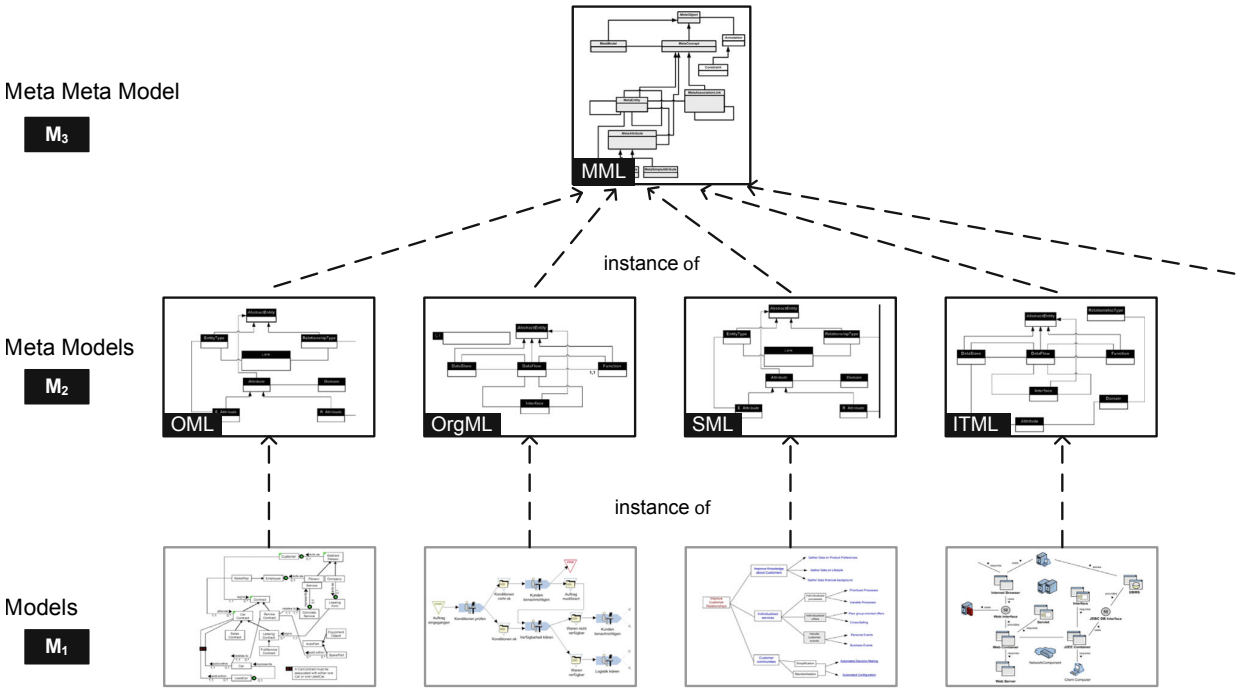

Figure 2: MEMO Language Architecture (Frank 2014)

relative to O_0, but *UniversityOfMannheim* with a `potency` value of "0" cannot be further instantiated at the next ontological level (O_3). The same approach is used for `Attributes` of `Clabjects`, i. e. their influence can be controlled using the properties called `durability` and `mutability`. While `durability` has the same meaning as the `potency`, the `mutability` states how often the value of an `Attribute` can be changed with respect to the instantiation of the `Clabject` at different ontological levels.

3 Multi-Level, Enterprise Architecture Modeling

The advantages of multi-level modeling for EA modeling have been most clearly articulated and demonstrated by Frank (1994), who have recently evolved their MEMO EA modeling method into a fully-fledged multi-level modeling approach, implemented using the XModeler tool (Clark and Willans 2013). The key new piece of technology that makes this possible is a special, multi-level-aware meta-meta-model, called FMMLx (Frank 2014). As shown in Fig. 2, which illustrates the new MEMO environment's language architecture, FMMLx is the top level model. This can be instantiated to define further languages at the meta-model level (M_2) which, in turn, can be instantiated to create user models at the M_1 level below. The data at run-time, normally shown within models, exists at the M_0 level which is instantiated from M_1.

The key differences to a classical model stack, such as the UML infrastructure, are that (a) the number of levels can be extended as needed to best represent the domain in hand and (b) the same concepts are usable in the same way at all levels. Thus, modelers do not need to resort to different concepts to represent instantiation at different levels (e. g. stereotypes versus standard instantiation). This, in turn allow users to define new languages (e. g. view types) as easily as they can use languages to model domain content.

MEMO predefines, out-of-the-box, several domain-specific modeling languages. At the time

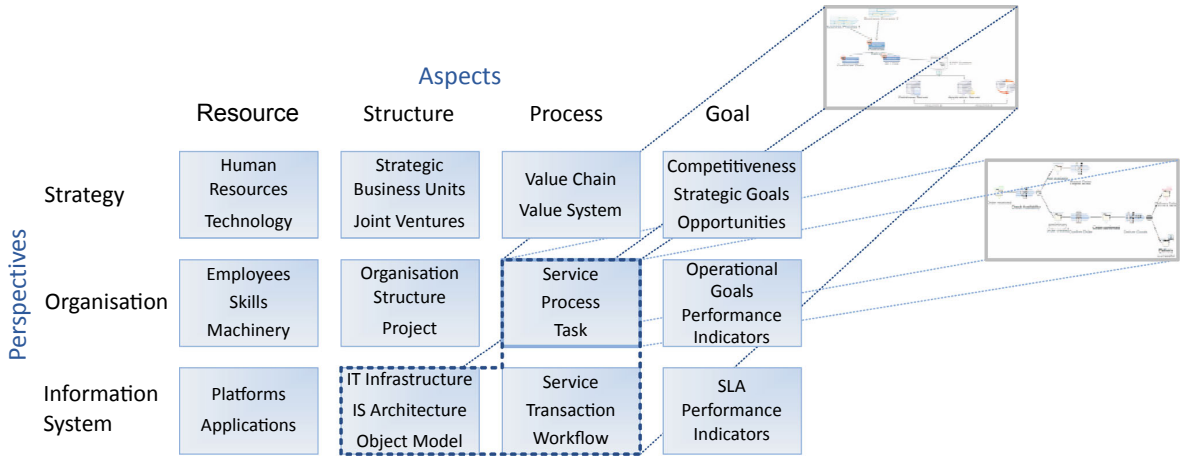

Figure 3: MEMO High-level Framework (Frank 2014)

of writing, these are: the strategy modeling language (MEMO-SML) (Frank 2002), the organization modeling language (MEMO-OrgML) (Frank 2011), the object-oriented modeling language (MEMO-OML) (Frank 2002), the organizational goal modeling language (MEMO-GoalML) (Bock and Frank 2016) and the IT infrastructure modeling language (MEMO-ITML) (Kirchner 2008). In addition there are more specific languages to describe indicator systems (MetricML) (Strecker et al. 2012) and decision processes (DecisionML) (Bock 2015). These languages define the view types that can be used to portray information about the system or organization in question. As illustrated in Fig. 3, these are organized as a two dimensional matrix based on the *perspective* they offer and the *aspects* they convey. For instance, business process models are assigned to the *perspective* "organization" and the *aspect* "process", whereas structural descriptions of the enterprise are assigned to the *aspect* "structure" and a value of the *perspective* dimension according to the needed abstraction level. As illustrated in Fig. 3, the cells of MEMO's matrix, which essentially constitutes its viewpoint framework, contain (or refer to) views (i. e. models) which are expressed using one of the languages mentioned above. MEMO also allows views to be mapped to combinations of cells in order to support models which span multiple perspectives and/or aspects.

Fig. 4 shows an example of the use of the MEMO Framework. In the example, two languages of the framework, MEMO-SML and MEMO-OrgML, are used to model aspects of an Insurance Brokerage company. In terms of MEMO's viewpoint framework, the presented example is situated in the cells represented by "strategy / process" and "organization / process". The view corresponding to the first cell uses the MEMO-SML language to describe an excerpt of a strategy model in the example that shows a value chain with one activity group being decomposed into further activities. Similarly, the view corresponding to the second cell uses the MEMO-OrgML language to model business processes which are part of an organization model. In order to ensure the overall consistency of views covering all cells shown in the viewpoint framework, the concepts from the different cells refer to each other. In the shown example the boundary between the two cells is the relationship of the `Activity` and the `Business Process` model elements. The activities coming from the strategy model are related to one or more business processes which describe the activities in the organizational model. This provides clear traceability from the strategic concepts down to their realization on the organizational and technical levels.

The key property of the MEMO modeling architecture shown in Fig. 2 is that all the levels are

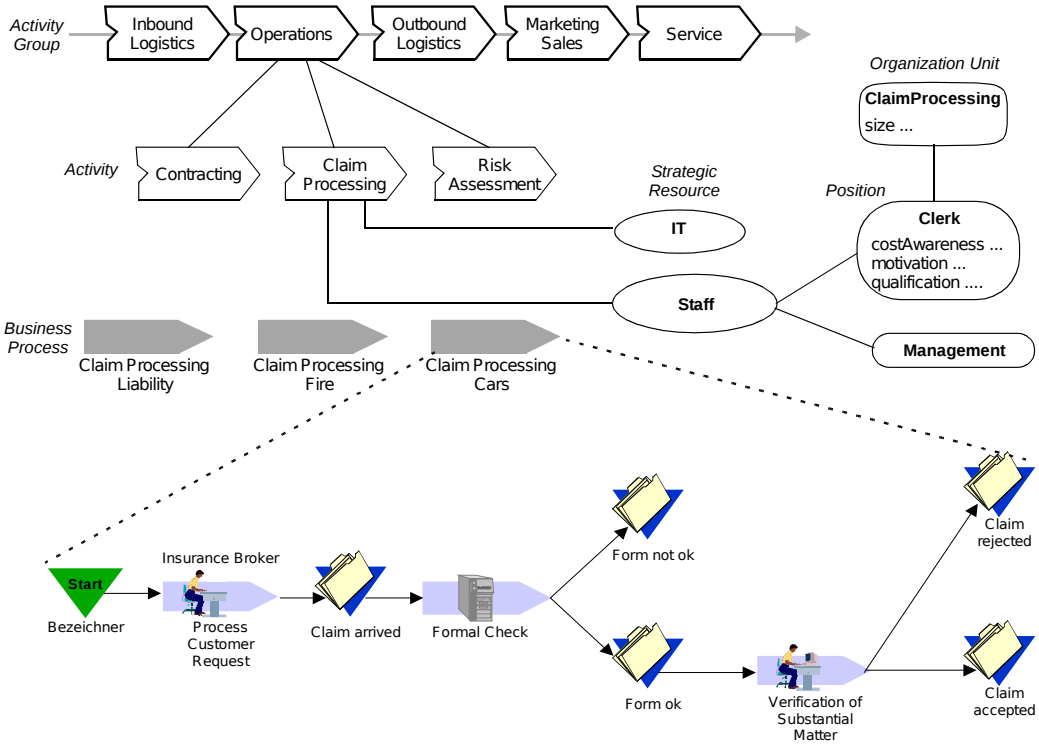

Figure 4: MEMO Example (Frank 2002)

"soft" in the sense that they are immediately accessible and changeable without transformations or code generation. In fact, from the point of view of the underlying tools, all levels are just data. MEMO does not focus on supporting views of operational information but the multi-view modeling framework makes it easy to do so. To support operational information in a seamless way a further layer, M_O, would need to be added containing instances of the models at level M_1. To do this efficiently, of course, MEMO would need to be extended with additional languages to describe such things as configuration and operational history information, and ultimately to support the definition of executable models (i. e. code). However, the basic capability for seamless extensibility and deployability is provided by the underlying multi-level modeling infrastructure.

4 Orthographic Modeling

Although MEMO represents a significant step forward over existing EA modeling approaches in terms of its ability to support seamless, multi-level modeling, and through this a seamless transition from development to operation (in the sense of DevOps), it is much more traditional in terms of its viewpoint framework. We believe that, to create the ideal foundation for MDO, it is necessary to integrate the benefits of multi-level modeling with a new kind of viewpoint framework that provides a more systematic and intuitive way of organizing and navigation around views. In this section we first motivate the need for such a vision and then explain the basic idea behind our approach which we refer to as "orthographic modeling".

4.1 Multi-View Modeling Realization Strategies

Although EA modeling approaches agree on the use of multiple views to describe an enterprise architecture, there is no consensus on how these views should be organized and supported. Atkinson et al. (2015) present a number of dichotomies that characterize the range of fundamental design

choices available for multi-view specification environments, including EA modeling approaches, based on the existing literature and state-of-the-art. The most important of these are summarized below.

Rigorous versus Relaxed. This is the most basic dichotomy which basically characterizes the level of formality and prescriptive guidance provided by an approach. "Relaxed" approaches such as Zachman and TOGAF provide few if any rules about what specific languages should be used to represent different views and how they should be populated, while "rigorous" approaches such as Archimate, RM-ODP and MEMO provide strong constraints on these issues. Except for extremely small organizations, relaxed approaches are unable to provide the control and discipline needed to support a fully-fledged MDO.

Synthetic versus Projective Views. One of the most fundamental design choices when realizing a multi-view approach is whether views are "synthetic" or "projective". Although the term was coined by some of the earliest work on multi-view approaches (Finkelstein et al. 1992), this terminology was popularized in the IEEE1471/ISO42010 standard for Systems and Software Engineering – Architecture Description (ISO/IEC/IEEE 2011) which defines the difference in the following way: *"In the synthetic approach, an architect constructs views of the system-of-interest and integrates these views within an architecture description using model correspondences. In the projective approach, an architect derives each view through some routine, possibly mechanical, procedure of extraction from an underlying repository."*

Projective approaches therefore revolve around a repository that stores a representation (i. e. a model) of the system from which the views are generated on demand by an automated transformation. The term Single Underlying Model (SUM) (Atkinson and Draheim 2013; Atkinson et al. 2011) is often used to refer to this repository as it is conceptually a single, complete and high-fidelity model of the real system. Most EA modeling approaches, including MEMO, do no explicitly explain whether they are synthetic or projective, leaving the choice open to individual tools. The exception is RM-ODP (ISO/IEC/ITU-T 1997) which is explicitly based on, and strongly advocates, the synthetic approach. The big problem with synthetic approaches is that inter-view consistency has to be maintained on a pairwise basis. This becomes untenable for large MDOs since the number of inter-view consistency relationships that have to be maintained grows exponentially with the square of the number of views.

System-Centric versus Component-Centric Views. Another important property of multi-view modeling approaches is how the subject of views is characterized – using a system-centric strategy or a component-centric strategy (Atkinson et al. 2015). In the former all views are characterized (i. e. identified) as being views "of" the same subject – the system. This means that the viewpoints, and thus the viewpoint framework, are determined only by the view types. In the latter, all views are characterized (i. e. identified) as being views of a distinct subject, either the system, or a component (i. e. a part) of the system. This means that viewpoints are determined not only by a view type but also by a view subject.

All mainstream EA modeling approaches today, including MEMO, support system-centric views. Of course, users of approaches based on synthetic views invariably create models that only describe a part of the system (e. g. a server, a process, a department etc.) since it is usually impossible to create views of the whole system. However, when doing so they have to go outside the viewpoint framework and use ad-hoc techniques to characterize what a view is describing. This in turn leads to numerous problems, including duplication of information, confusing characterization of views and the lack of guidelines for filling them with content (Atkinson and Tunjic 2014b).

Abstract versus Concrete. Another important design issue in EA modeling approaches is whether views are essentially "abstract" (i. e. logical) concepts that have no direct representation, "concrete" (i. e. physical) concepts that have a physical representation (e. g. on a computer screen or in a printed document) or a mixture

of both (Atkinson et al. 2015). The difference revolves around whether the views are intended to correspond to individually "viewable" chunks of information that can be seen in one go on a computer screen or in a document, or a more loosely-related collection of model elements that cannot conveniently be viewed in one piece. The former kind of view is often referred to as a "diagram", while the latter is often referred to as an "abstraction level" or "perspective".

None of the existing EA approaches make their position on this issue explicit, but in practice they invariably adopt just one of the approaches to the exclusion of the other. Because of the high level of granularity and abstraction at which they operate, the views of relaxed EA modeling methods such as Zachman and TOGAF are abstract. The same is true of the RM-ODP method, however, which despite being rigorous has very large grained views that cover information from a particular high level perspective. According to RM-ODP, the creation of concrete diagrams is a tool issue which lies beyond the method itself. The views of most other rigorous methods such as Archimate and MEMO are concrete since they are defined using a prescribed language and are intended to be rendered for physical representations. In practice, to model large scale approaches, a mixture of both kinds of views is necessary. Ideally it should be possible to define both abstract and concrete views using the same metaphor and to allow the latter to be nested arbitrarily inside the former. We therefore refer to this as the requirement for "composite views" since regarding abstract and concrete views as being leaves in the composite pattern (Gamma et al. 1995), and thus being arbitrarily nestable, provides the perfect model.

In fact, the users of all existing methods already have to learn to work with both kinds of views because abstract views have to be broken down into smaller, individually viewable "models" (i. e. diagrams), while concrete views have to be organized in some way into larger, cohesive bodies of information. The problem with all existing EA methods is that one or other of the two forms of views (i. e. abstract or concrete) is implicit and has to be handled by users in an ad-hoc way outside the framework of the method. In the case of MEMO for example, what the method calls "views" are concrete views, while what the methods calls "perspectives" or "aspects" are abstract views.

4.2 Orthographic Software Modeling

As argued in the previous section, none of the well known approaches to EA modeling provides the ideal combination of realization choices to support the view-based modeling of large systems and organizations. The goal of the Orthographic Software Modeling (OSM) approach proposed by Atkinson et al. (2010) is to support such a combination in an efficient and highly intuitive way by appealing to the successful notion of orthographic projection used in CAD tools for engineering physical artifacts. This is illustrated in Fig. 5. The left-hand shows orthographic projections of a physical object (a house) while the right-hand side shows orthographic projections of some abstract entity that is not physically visible. As implied by its name, OSM was originally focused on the orthographic modeling of software, but in general the cloud in the middle of the figure can represent any well defined conceptual or physical object, including complete IT systems or socio-technical systems such as organizations.

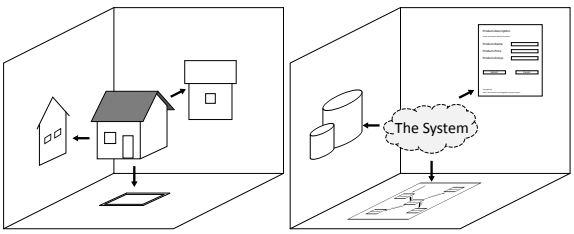

Figure 5: Orthographic Projection

The orthographic projection metaphor inherently suggests the realization choices described in the previous subsection. First, the approach has to be *rigorous* since the rules for determining the content of a view once the viewpoint is known have to be completely unambiguous. Second, the very idea of orthographic projection calls for individual views to be thought of as *projections* from

the underlying artifact (i. e. a SUM) rather than as artifacts in their own right from which the properties of the viewed object are derived. Third, since it is practically impossible to include all information about a large system in a single concrete view, it is convenient to employ *component-based* views which "zoom in" on one part of the system (e. g. a particular room or feature of a house). Fourth, since such fine-grained views need to be though of as belonging to larger-grained views (e. g. a front view of the door is a part of the front view of the house) all views need to be *composable* to arbitrary depths.

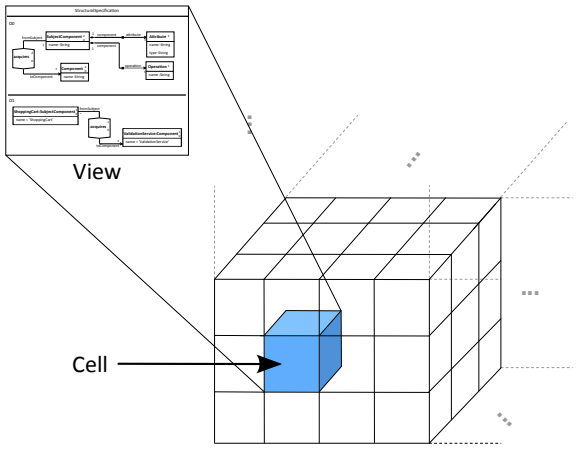

Figure 6: View Selection based on Dimensions

Previous papers on OSM have characterized the challenge of building an OSM environment as having three main ingredients (Atkinson et al. 2010). The first is to identify a suitable *dimension-based* metaphor for identifying and navigating around views of a logical subject rather than a physical object. Obviously the normal dimensions of the real world are not suitable for software systems or organizations. This idea is shown schematically in Fig. 6. All views are identified and conceptualized as existing within a multi-dimensional space and are selected by picking the appropriate coordinates. Concrete views correspond to individual cells or small combinations of cells, while abstract cells correspond to large collections of cells that reflect a slice or sub-cube of the dimension space. For example, in a viewpoint framework containing a dimension called *Platform Independence*, all the concrete views that share the value *PIM* in this dimension (but have different values for other dimensions) can be regarded as making up an abstract view corresponding to a "platform independent model" of the system. Moreover, all views are inherently identified by the subject they are portraying as well as the properties they are displaying.

The second ingredient of OSM is the "on demand generation" generation of views from a *Single Underlying Model* (SUM) which holds all information concerning the system under development. The SUM has no visual representation and is never accessed directly by the user so it can only be seen and manipulated through the views. Since they are generated automatically, on demand, such views naturally represent projections of (parts of) a system. The consistency between the views is guaranteed by their continuous synchronization with SUM. This principle is shown schematically in Fig. 7.

The third core ingredient is a *view-based method* which inherently promotes the use of multiple dimensions and views to represent a system. The initial software engineering-oriented version of the OSM approach adopted the KobrA (Atkinson 2002) method for this purpose. However, to support the MDO vision through orthographic modeling a more general method (i. e. definition of views and dimensions) is required.

5 Deep Orthographic Modeling

As explained in Sect. 3, the MEMO method has pioneered the use of multi-level modeling technologies to seamlessly integrate development-time (i. e. architectural) views and run-time (i. e. operational) views of organizations to facilitate the MDO vision. However, its viewpoint framework for identifying, characterizing and navigation around views is rather traditional. On the other hand, "classic" OSM proposes a new metaphor for addressing the latter requirement which naturally supports component-centric and composite views, but does

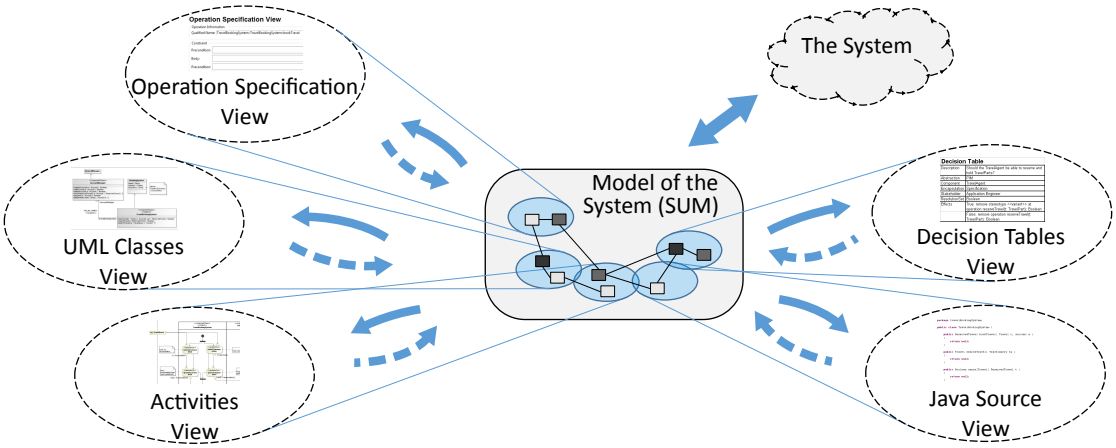

Figure 7: SUM-based Projective on Demand View Generation

nothing to address the seamless integration of architectural and operational views. The contribution of this paper is therefore to combine these two approaches, the deep variant of multi-level modeling and orthographic modeling into a single unified approach for enabling the MDO vision. The motivation for attempting this unification is the expectation that (a) the two approaches are naturally compatible and will deliver a powerful synergy, and (b) the unified, deep orthographic modeling approach will provide the best platform for supporting the flexible, view-based modeling metaphor needed for realizing the MDO.

Essentially, deep orthographic modeling uses the same principles as "classic" orthographic modeling, but uses deep modeling technology for the underlying storage and model representation platform. This is illustrated schematically in Fig. 8. In order to fully exploit the capabilities of deep modeling we apply the technology also on the views, which means that the SUM, the projected views and the projections of the views are based on the deep modeling technology. Thus each of the rectangular elements in the figure are intended to represent instances of the OCA shown in Fig. 1. The thin orange strip along in each OCA rectangle is mean to represent the linguistic meta-model in the L_2, while the large grey section of each OCA rectangle is meant to represent the domain content distributed over an arbitrary number of ontological levels.[1] Note that not only the SUM and the view are in general deep models, but also the transformations (i. e. rules and traces). Notice also that all of content in the SUM is "greyed out" to convey the idea that it cannot be directly seen and has not concrete syntax. In contrast, one or more of the ontological levels in the views are highlighted in color to convey the idea that they are physically rendered using a concrete syntax.

The following sections present the main conceptual ingredients and prototype realization of such a deep orthographic modeling environment. To support the concepts presented in the next sections, we use UML class diagram like notations to describe the environment configuration and provide a kind of construction kit. The definition of the construction kit will be modeled in the M_1 layer, while the M_0 layer will be used when the environment is used to specify a concrete project.

5.1 Deep Single Underlying Model

The SUM plays the central role in an orthographic modeling environment since it serves as the place where all known information about the system or organization in question is stored. "All information" means the data which is needed to provide a detailed, precise and full description of the system

[1] Although only three levels are hinted at in Fig. 1, since this is a highly schematic diagram, in general, the number of levels is flexible.

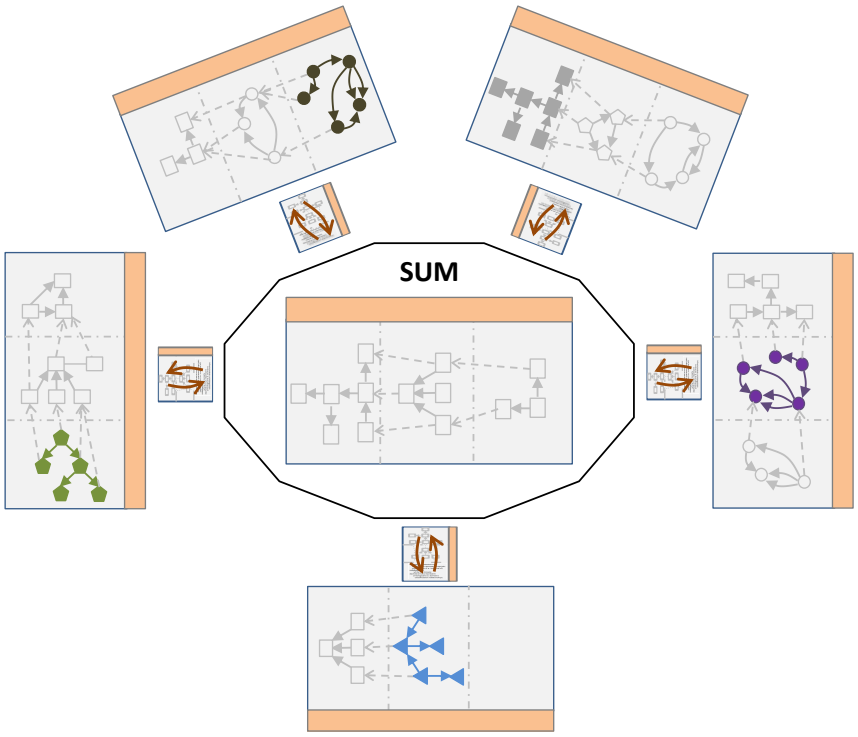

Figure 8: Architecture of Deep Orthographic Modeling (SUM, Views and Projections realized using Deep Modeling)

of interest in the form of a model. The data can range from the system's behavioral properties and feature specifications to its architectural composition and executable descriptions (and ultimately, running instances). In the ideal case the real system can be derived directly from the SUM, or "is" the SUM. The SUM should also ideally be redundancy free, so a given piece of information about the system is only stored once.

Architects who are working on the SUM to describe a system work with projective views. This means they never directly "see" the SUM, so it does not need to support bindings to concrete syntaxes and can be optimized for storing information efficiently. This allows the SUM to be defined as a redundancy free model using the "information compression" and "information expansion" approaches described in (Atkinson et al. 2015). By the term "information compression" we mean the process by which information belonging to many concepts in the SUM is compressed into fewer concepts in a view. This is used in views which provide some kind of overview by aggregating information similar to OLAP views. By the term "information expansion" we describe the process by which information from many views (or many model elements of one view) is stored in one model element in the SUM. This can be applied to conceptual concepts that are relevant across many abstraction layers (e. g. *organization* and *information system* perspectives of MEMO). The common concepts of the views exist only once in the SUM, but can be seen multiple times in different views by expanding the single SUM representation. This approach is practicable since the SUM does not care about the visual representations of the compressed and expanded concepts. The visual representations of concepts are handled outside the SUM when they are projected into views and are visualized according to the rules and language of each view.

In order to capture all relevant information about the system under discussion, the SUM must be sufficiently expressive. The concepts which can be used in the SUM to store user data are defined in the most abstract ontological level (the meta-meta level) of the SUM. The key question, therefore, is which concepts are needed in the SUM to provide the needed expressiveness to capture sufficient information for a detailed and full specification of a system. This challenge is addressed by the methodologist who, in orthographic modeling, has the role of setting up the environment by defining the ontological meta-model of the SUM, the needed views and the dimensions used to navigate around the available views. In short, the methodologist is responsible for setting up the framework used to specify the system according to a specific method.

For a software system, a method such as KobrA (Atkinson 2002) could be used, while for EA modeling a method such as Archimate or MEMO could be adapted. Based on the choice of method, the methodologist can provide a configuration for the orthographic modeling environment by defining the needed artifacts. The views and dimensions for the orthographic modeling environment can be derived from the chosen method. For KobrA, natural views are *structural*, *behavioral* and *functional*, as defined by the method. Since KobrA supports the model-driven development approach, it also makes sense to reflect the *platform independence* of a view in the definition of the dimensions. For the MEMO approach, suitable candidates for dimensions are the two concerns which define the MEMO high-level Framework as shown in Fig. 3).

When used for a view-based, model-driven organisation, the SUM is a model containing all known information about the organization of interest as it can be seen in Fig. 7. When represented as a deep model, the SUM can represent information at all levels of classification seamlessly, using the same notation and concepts. This in turn, means that operational data, which typically occupy the lower ontological levels, and architectural data, which typically occupy the higher ontological levels, are accessible seamlessly using the same language and conventions. The most abstract level in a deep model (i. e. O_0) contains the (domain) meta-model for the information shown in the views. The levels below (i. e. $O_1 \ldots O_n$) contain descriptions of the organization at different levels of abstraction and classification (i. e. architectural and operational). The number of ontological levels for a particular specification depends on the domain and the scenario in hand. Moreover, the number of ontological levels and types storeable in the SUM is not limited and can easily be extended as needed without recompilation and redeployment of code.

When a deep modeling infrastructure is used to store the SUM, the contents of the SUM are represented using the linguistic concepts of the deep modeling language (L_2). In the presented approach, the structure of the SUM follows the OCA as shown in Fig. 8. Since the SUM should ideally have a predefined ontological meta-model that defines the types used to capture user data, we describe the SUM using two distinct parts – the predefined part which we call the "SUM language" and the evolving part containing user data which we call the "SUM content". The SUM language is usually contained in the top ontological level(s) (i. e. O_0), while the SUM content is contained in the subsequent ontological levels. The language and the content parts can be seen as two disjoint sets of model elements.

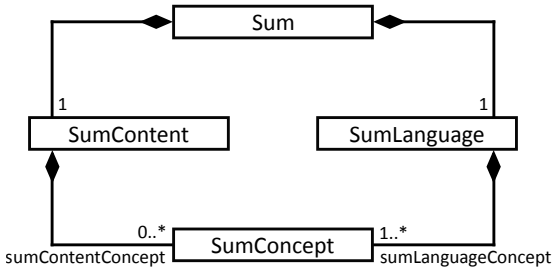

Figure 9: Structure of the Deep Model SUM

The structure of the SUM is shown in Fig. 9 as a UML class diagram. The `Sum` model element contains the `SumLanguage` and `SumContent` elements. Both are composed of elements of the type `SumConcept` which is used as a placeholder

or pointer element for any concept of the deep modeling language (e. g.`Clabject`, `Attribute`, `Level`, `Classification`, ...). For example if the deep model from Fig. 1 were the SUM, the `SumLanguage` would contain instances of `SumConcept` pointing to the ontological level O_0 and the `Clabject` *OrganizationType*. The `SumContent`, in turn, would contain instances of `SumConcept` pointing to the following concepts – the ontological levels O_1 and O_2, the *University* and *UniversityOfMannheim* `Clabjects` and the corresponding `Classification` relationships which classify *University* as an instance of *OrganizationType* and *UniversityOfMannheim* as an instance of *University*.

The `SumLanguage` must have at least one `SumConcept` in order to enable the creation of ontological instances to store user data. However, the `SumContent` can be empty. This is the case, for example, when the `Sum` does not yet contain any user data at the start of a new project.

5.2 Deep Projective Views

Views are the user interfaces in view-based environments. They must be integrated in a way that ensures the consistency of the distributed information used to specify a system. In a projective approach, consistency must be ensured between the available views and the SUM since this ensures the overall consistency of all the views. The views are automatically consistent with one another if they are individually consistent with the SUM.

The projective approach implies that views are projections of the SUM, i. e. they show particular parts of the SUM which describe the system under development. But how many views are needed to specify a system and what content should the views have? The orthographic modeling environment must have sufficient views to allow architects to describe every relevant aspect and part of the organization in question.

When defining the views in an orthographic modeling environment to support a particular method the methodologist must first define the view language for a view. The view language is the predefined part of a view which describes the domain concepts represented in the view. In other words, it defines the concepts which can be used to embed user data and thus information about the organization into the views. The view language plays the role of a meta-model in classical two-level modeling and contains the types which can be instantiated in order to capture user data. After defining the view language the methodologist must define how the view is projected from the SUM, and vice versa. This step includes manipulations of the SUM, that is – definition of types in the SUM used in the view and the definition of concrete relationships between the view and the SUM.

The relationship between a view and the SUM in the context of orthographic modeling is shown in Fig. 10. The *view language* is the static predefined part of a view and the *projection rules* are the static predefined rules which relate types from the view language to corresponding types from the SUM language. The *view content* is the dynamic content of a view which contains user data from the SUM. The *projection traces* are derived according to the projection rules and capture the relationship between the user data from the SUM and the corresponding user data presented in the view. The set of types from the SUM language that are projected to the view language is the *scope* of the view. The view scope belongs to the static part of a view. The *view footprint*, on the other hand, belongs to the dynamic part of a view since it is the user data from the SUM which is mapped to view content by the projection traces.

The following subsections present the structure of views in orthographic modeling and their relationship to the SUM.

5.2.1 View Language and View Content

In order to provide stable projections of views from the SUM, we divide views into two parts – the view language and view content. The former corresponds to the notion of a meta-model in classical two-level modeling, but in our case the view language can span more than one ontological level. The latter corresponds to the model in classical two-level modeling environments, which

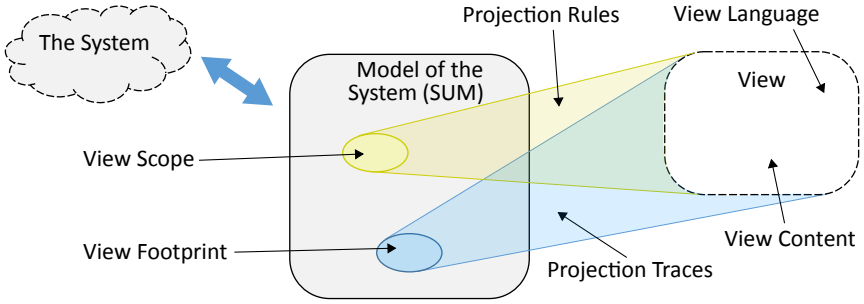

Figure 10: Projection-based Relationship between SUM and View

is an instance of a meta-model. Again this can also span multiple ontological classification relationships, if desired. The two parts are separated since the projection rules must be defined on the types in the language part of a view and the SUM. In contrast to the content part, the language part is static from the perspective of an architect using the orthographic modeling environment to describe an organization.

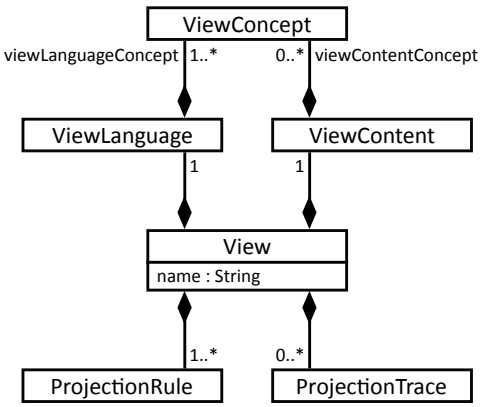

Figure 11: Structure of a View

The concrete structure of a view is shown in Fig. 11. The `ViewLanguage` element contained by the `View` element contains the types or the meta-model of the view. The `ViewContent` part of a `View` is a container for the concepts that represents user data in the view. The `ProjectionRule` part of a `View` describes how the projection is realized as a transformation – i. e. it identifies which concepts from the SUM are projected to the view. While the `ProjectionRules` define the projection on the type-level, i. e. on the meta-model, the `ProjectionTraces` hold the mappings of the concepts on the instance-level. The `ProjectionTraces` of a view are automatically generated by the orthographic modeling environment as soon as a view is projected. The `ProjectionRules` are used to query the user-data from the SUM and show it in the view. During this process the `ProjectionTraces` are generated and assigned to the `View`.

The view language and content parts of a view are shown in the upper part of Fig. 11. The `ViewConcept` element is used in the same way as the `SumConcept` element, described in the previous section. The `ViewConcepts` which belong to a `ViewLanguage` are contained in the `viewLanguageConcept` relation of the `ViewLanguage`. In a view language there must be at least one `viewLanguageConcept`. In a similar way to the view language, as shown in Fig. 11, view content is represented by the `ViewContent` element contained by a `View`. The `ViewConcepts` which belong to a `ViewContent` are contained in the view content's relation `viewContentConcept`. As in the SUM, the data in the view's content part can contain information about an organization at different levels of abstraction and classification (i. e. architectural and operational).

To realize our approach and exploit the benefits of a deep modeling infrastructure, we use the level-agnostic modeling language (LML) defined by Kennel (2012) and implemented in the MELANEE tool by Gerbig (2017), based on the Eclipse Modeling Framework (EMF) (Budinsky

et al. 2003). Beyond the ability to handle the ontological levels dynamically, MELANEE provides advanced features for visualizing deep models in multiple formats and notations (Gerbig 2017). These employ visualizers to influence the appearance of the concepts in the deep model. These visualizers, of which there can be many at the same time, can be customized by the user at run-time to provide the most suitable concrete syntax for each specific holder. Moreover, users can toggle between domain specific visualizations and general purpose visualizations of models or model elements at any time. For instance a BPMN like deep model (OMG 2011a) can have two domain specific visualizations defined, one presenting the concepts using the regular BPMN symbols and a second presenting the concepts using a general purpose notation (e. g. UML like). While the former is likely to be preferred by BPMN experts, the latter is probably more accessible to BPMN beginners who are not familiar with the graphical notation.

The visualizers are defined in the context of the domain concepts, which means that every concept can have its own visualizer. If no domain specific visualizer is defined for a concept its instances will be rendered using a default general purpose visualization. The general purpose visualization renders entities (i. e. classes or objects) using the usual rectangular notation from the UML, and connections using the flattened hexagon notation as used in entity-relationship diagrams. Furthermore connections can be collapsed into lines to save space in a diagram. A strength of MELANEE is its capability to use the visualizers of a domain specific visualization across many ontological levels based on the classification hierarchy. A visualizer defined for a concept at O_0, can be automatically used for its instances at all the ontological levels below ($O_1 \ldots O_n$).

5.2.2 View Projection

This section describes how information in the SUM is projected into a view. The transformations that enact the projection process are basically defined at the level of the types in the SUM and view languages. Defining the projections at the type-level makes the views generic since they can be applied to specific parts of the same as it evolves. In other words, this approach supports the notion of component-centric views that portray a particular component.

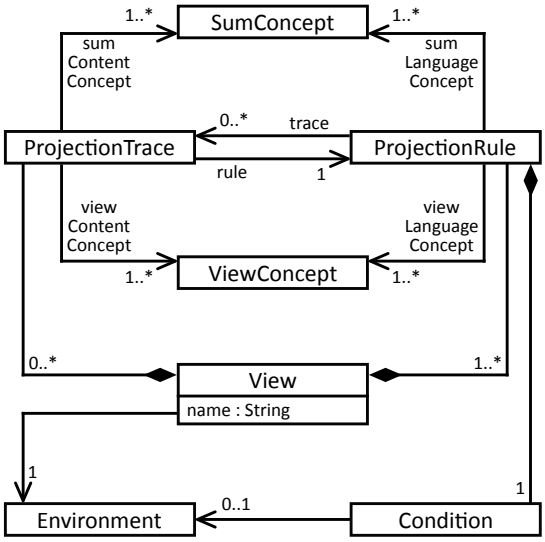

Figure 12: View Projection Structure

Fig. 12 shows the structure of a view projection. The figure summarizes the concepts from the previous sections and extends the structure for the orthographic modeling environment.

Projection Rules

A projection rule is defined as a relation between the SUM and a view relating one or more types from the SUM with one or more types from the view. This is controlled by a condition expression which can constrain the projection in order to project a specific set of concepts. The `ProjectionRule` element from Fig. 12 therefore has references to the language concepts from the SUM and the view (`sumLanguageConcept` and `viewLanguageConcept`) and it contains an element of type `Condition` which is used to provide fine-grained control to the projection rule. Since the projection rules are defined on the types, they are applicable to all instances in a specific projection. Using the `Condition`, a projection rule can be configured to project a chosen subset of all the instances of the types. This feature allows the re-use

of view definitions – i. e. the methodologist defines one view (-type) which can have many entities (-instances) in an orthographic modeling environment. The input to the `Condition` is derived from the orthographic modeling environment, or more specifically, from the dimension-based navigation mechanism (cf. Sect. 5.3). This mechanism allows on-the-fly view generation based on the content of the SUM which is used as a parameter in the projection rule (Atkinson and Tunjic 2016).

The ability to parameterize a view's projection definition follows the idea of component-centric views, as presented in Sect. 4, and plays an important role in the generation of *subject*-oriented views, which are views of a specific part of the SUM and are used in Sect. 5.3, for the definition of the dimension-based navigation approach. The projection of a view from the SUM is usually achieved using multiple projection rules. For example, there is often at least one projection rule for each type.

Projection Traces

The views generated by a projective approach contain user data derived from the SUM. Projecting user data from the SUM into a view populates it with user data according to its projection rules. Since the view is derived from the SUM it cannot contain any user data which is not in the SUM. The views in our approach are therefore always abstractions of, or windows onto, the SUM. The transformation of the information from the SUM to the view gives rise to projection traces which map the concepts in the view content with the corresponding concepts in the SUM content. Projection traces are created when projection rules are applied to project (i. e. transform) user data from the SUM into a view. A projection rule can lead to many projection traces, but may also lead to no projection traces – this is the case when the types from the SUM referred to in the projection rule have no instances. The ability, to retrieve the mappings of the concepts on the type and instance-level allows the definition of rules which can be used for synchronization mechanisms between the SUM and the views in order to ensure the consistency of all view. This is realized using lens-based technologies developed by Foster et al. (2007) as explained by Tunjic and Atkinson (2015).

Fig. 12 shows the structure of the projection trace elements. A `ProjectionTrace` results from the application of a `ProjectionRule` and is assigned to it via the `rule` relation. In contrast, a `ProjectionRule` can have many `ProjectionTraces` assigned to it, since a single `ProjectionRule` can result in the projection of many instances of its defined types and thus create many `ProjectionTraces`. The concepts from the view and the SUM which are "traced" by a `ProjectionTrace` are referenced via the `sumContentConcept` and `viewContentConcept` relations. The projected `View` contains information about the alignment of the concepts on both the type and instance-levels and thus contains the `ProjectionRules` and the `ProjectionTraces`.

5.3 Dimension-based View Navigation

In view-based modeling, every view should exist for a reason. In the IEEE1471/ISO42010 standard, the existence of views is based on concerns of the stakeholders, so every view takes at least one concern into account. The problem for architects when dealing with many views is to find the right view for their needs. The search for the right views can be guided by concerns, but since concerns are predefined and static, the views which can be derived from them are predefined and static as well. To be able to support suitable views, whose contents are defined dynamically as described in Sect. 5.2.1, we use a multi-dimensional cube (hyper-cube) metaphor for navigating around views. The approach has some similarities to the OLAP navigation model known from the data warehouse domain (Kimball and Ross 2013) but goes beyond it by mixing operational views (primarily at the lower, more concrete ontological levels) and static views (primarily at the higher, more abstract ontological levels).

Our approach is a mixture of the two extremes mentioned above. To define a view, we use predefined static parameters and to describe what the

view should be of, we use dynamic parameters. Both parameter types are mapped to dimensions which span the hyper-cube. The available views are contained in the hyper-cube and are situated in cells – one for each view. A cell, and thus a view, is selected by picking values for each dimension.

Figure 6 shows the idea of the hyper-cube, in which a view is assigned to a cell. The content and type of the view are influenced by the dimension values of the cell in which the view is located.

5.3.1 Hyper-Cube Definition

This section provides an overview of the structure of the hyper-cube approach. As shown in Fig. 13, a `Cube` contains many `Dimension` elements which span the hyper-cube, with the number of dimensions determining the order of the cube. The Zachman Framework (Zachman 1987), for example, can be realized with two dimensions. The dimensions which span a cube can be *static* or *dynamic*. A cube must have at least two dimensions – one static and one dynamic. The `Dimension` model element is therefore defined as abstract, while its sub-classes (i. e.`DynamicDimension` and `StaticDimension`) can be used to create concrete dimensions. Both dimension kinds are described by a unique name.

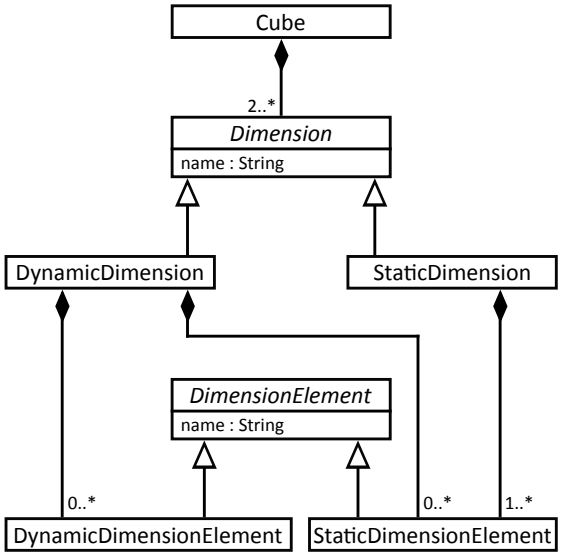

Figure 13: Hyper-Cube Structure

The dimensions of the cube, as well as the views themselves, should ideally be orthogonal to each other with minimal overlap. The dimensions contain elements of the type `DimensionElement` which represent the values of the dimension. To allow dimension elements to be either *static* or *dynamic* the `DynamicDimensionElement` and `StaticDimensionElement` elements are defined as specializations of the abstract class `DimensionElement`.

All values of all dimensions must have names. Static dimensions can contain only static dimension values and there must be at least one. They are used to describe the type of a view, since the view-type is also static and predefined. An example of a static dimension is *Platform Independence*, with its static dimension values *CIM*, *PIM* and *PSM* from the MDA specification (Belaunde et al. 2003).

A dynamic dimension consists of dynamic dimension values derived from the SUM. Additionally, a dynamic dimension can have further static dimension values. Examples of static dimension values in dynamic dimensions are: *none* and *all*. These can be used if the values of the dimension are not relevant for a view, or if all values are relevant for a view. A dynamic dimension can, in contrast to a static dimension, also be empty. This is the case when the dynamic dimension has no static dimension values defined and no suitable concepts for the dimension exist in the SUM. An example of a dynamic dimension is *Component*. The dimension values of this dimension would usually be the list of all available instances of the type `Component`. Using the aforementioned example of a static dimension, it is possible to select a platform-independent view of a component by choosing the value *PIM* from *Platform Independence* and the appropriate component from *Component*.

5.3.2 Deriving Values for Dynamic Dimensions

This section presents the mechanism used to derive the dynamic values of dynamic dimensions. This mechanism is an essential part of our approach

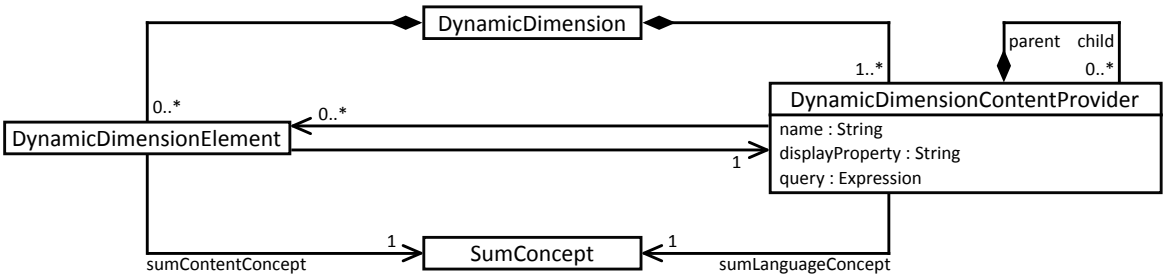

Figure 14: Dynamic Dimension Content Provider

and supports the usage of content from the SUM to create dynamic dimension values. Figure 14 gives an overview of the concept by extending Fig. 13. The parts of Fig. 13 which are not relevant for this section are omitted.

In order to populate a dynamic dimension with `DynamicDimensionElements` we use the concept of `DynamicDimensionContentProviders`. Dynamic dimension content providers are responsible for querying the SUM for information about particular concepts. The results are dynamic dimension elements which are assigned to the corresponding dynamic dimensions. A dynamic dimension must therefore have at least one dynamic dimension content provider, with a well defined name, which provides appropriate concepts for querying the SUM for relevant information using the type of the concept and a query expression. The type is given by the `sumLanguageConcept` relation resulting in a `SumConcept` from the SUM language. The `query` expression states which concepts should be queried. It is possible to accept all available instances, of the type given by `sumLanguageConcept`, or to filter it according to a particular condition. For example, "get all `Components` at the top level", means that all components that are not contained by other `Components` should be selected. The result of a query can be empty if no suitable concepts in the SUM exist. The `dynamicDimensionElement` relation provides all dynamic dimension elements which result from the considered dynamic dimension content provider. In order to get the `DynamicDimensionContentProvider` of a `DynamicDimensionElement` the relation `dynamicDimensionContentProvider` can be used.

The information from the SUM shown using dynamic dimension elements are instances of the type defined by `sumLanguageConcept`. Therefore the element `DynamicDimensionElement` owns the relation `sumContentConcept` which points to the `SumConcept` from the SUM content. The property `displayProperty` states which property of the `DynamicDimensionElement`'s `sumContentConcept` should be displayed as the dimension value. Usually the *name* property is used as the dimension value, so the dynamic dimension which contains all components contains all names of the components.

As shown by the *parent – child* relationship in Fig. 14, a dynamic dimension content provider can contain another content provider. This is the case when a hierarchical structure is needed within a dimension. This functionality can be used if, for instance, a dynamic dimension contains all instances of business processes and business processes can have sub-processes. A containing dynamic dimension content provider can be used to reflect this container-like structure in the dimension. Another example could be the representation of information which belongs together, e. g. the business processes and their instances. The business processes can be represented in the top hierarchical dimension while their instances would be represented in the next sub-dimension. In this case the top hierarchical dimension would contain the architectural data, while the sub-dimension would contain the operational data.

5.3.3 Views in a Hyper-Cube

In the previous sections we defined the structure and the behavior of the dynamic hyper-cube. In

this section we discuss the concepts which support the assignment of views to cells in the dynamic hyper-cube. To this end, we introduce the concept of defining a slice or sub-cube of the hyper-cube in order to assign views to it. The assigned views correspond to abstract views as defined in Sect. 4. Since a slice is just a special case, the term sub-cube will be used in the following. A sub-cube is defined when one or more of the coordinates that select cells are left undefined. This occurs, for example, at framework configuration time because the values of dynamic dimensions are no yet available. It is therefore not possible to identify a particular cell at this stage because it is not possible to select concrete values for all dimensions. Selecting concrete values for only some of the dimensions and leaving the other unspecified defines a sub-cube of the hyper-cube. Such sub-cubes correspond to the notion of abstract views as explained in Sect. 4.

Sub-Cubes

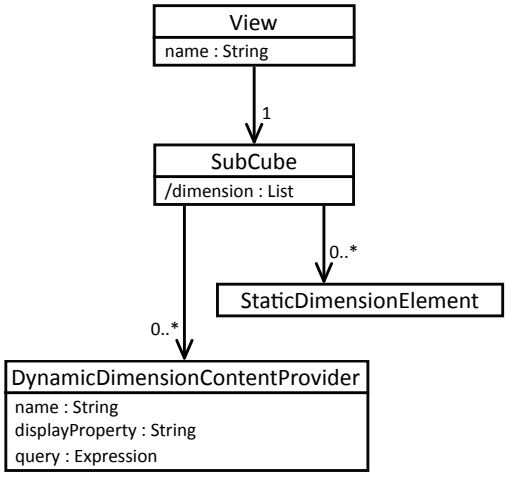

Figure 15: The Concept of a SubCube

Figure 15 shows the mechanism for defining sub-cubes of a hyper-cube. A sub-cube is defined by selecting a value for each existing dimension regardless of whether it is a dynamic or static dimension. The derived list property `dimension` contains every dimension that is available for the considered hyper-cube. The derivation is based on the relationships `dynamicDimensionContentProvider` and `staticDimensionElement`, since both belong to a dimension. The length of the list `dimension` must be equal to the number of dimensions in the hyper-cube and the content must be distinct.

The relationship `staticDimensionElement` allows the choice of static dimension values to define the sub-cube. The static dimension values can be contained by both static and dynamic dimensions. The relationship `dynamicDimensionContentProvider` allows dynamic dimension values to be chosen using the content providers. The values returned by the content providers depend on the content of the SUM and are thus not defined at method configuration time. Hence the sub-cube uses the content providers and not the concrete dynamic dimension values.

At configuration time the `Views` are assigned to the sub-cubes using the `SubCube` relation of `View`. Every view must be assigned to exactly one sub-cube in order to be usable in a project. The view, which is related to a sub-cube is an abstract view.

Figure 16 shows a view which is assigned to a sub-cube. The sub-cube is defined by the static dimension values *PIM* and *Structural* and a dynamic dimension content provider which returns all components. When a view is assigned to a sub-cube only the view language is available but not the view content, since the dynamic dimension values from the dynamic dimension content providers are not available at this point in time and the view content is thus empty. This is depicted in the figure by showing only the view language part of the view assigned to the sub-cube since the view content part is not available at method configuration time.

The static dimension values of a sub-cube describe the view language of the view which is assigned to the sub-cube. Thus, the view language of a view depends on the static dimension values.

From Sub-Cubes to Concrete Cells

In this section, the concept of a sub-cube will be extended to show how concrete cells are derived

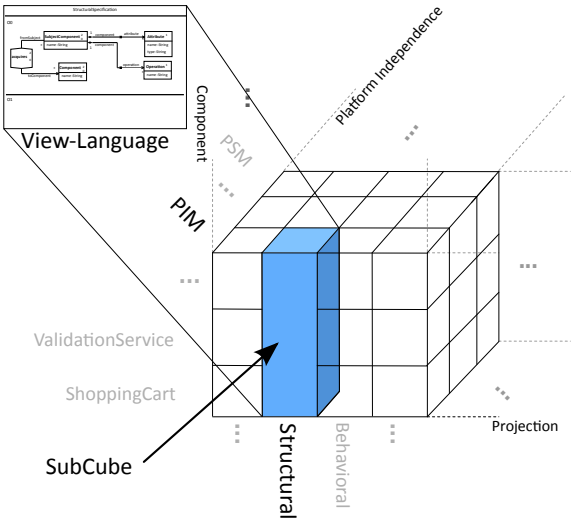

Figure 16: View Assigned to a Sub-Cube

from the previously defined sub-cubes. We therefore extend Fig. 15 and introduce the concept of a `Cell` which contains information about exactly one dimension value for each available dimension. The structure of a cell, and its relationship to the previously described concepts, is shown in Fig. 17.

The concrete cells are available as soon as a configured framework is used in a real project which must have a SUM from which the dynamic dimension values can be derived. In this step, the `DynamicDimensionContentProviders` generate `DynamicDimensionElements` based on their configuration. As soon as the `dynamicDimensionElement` relation of a `DynamicDimensionContentProvider` is not empty, the `SubCubes` which use the content provider can retrieve the actual dimension values for this dynamic dimension. Once this information is available, the `SubCube` can generate the concrete cells by using the concrete dynamic dimension values. The `cell` containment relation is used to get all `Cell` elements of a `SubCube`. The relation can also be empty when no suitable cells exist (e. g. when there are no components available in the SUM) so the corresponding cells do not exist.

The `Cell` elements have the same `staticDimensionElements` as their owning `SubCube`. The relation `dynamicDimensionElement` depends on the result of the content provider and contains one value for each `dynamicDimensionContentProvider` from the owning `SubCube`. The derived property `dimension` of a `Cell` must contain exactly one dimension value from each available dimension.

The relation `cell` of a `View` denotes the assignment of a view to a concrete cell, while the view corresponds to a concrete view. These relations are generated when the orthographic modeling environment is used to specify a system in a concrete project. At run-time, the content providers query the SUM and produce concrete cells. In this step, the `View` which is assigned to a `SubCube` gets assigned to all the generated `Cells` of that `SubCube`. The view content is generated at the moment when a `View` is assigned to a `Cell`. Then the environment parameters are used in the conditions of the projection rules, resulting in the projection of concepts from the SUM content to the view's content part. This procedure is triggered automatically as soon as the content of the SUM changes, i. e. as soon as the set of `dynamicDimensionElement` relations of a `DynamicDimensionContentProvider` used by a `SubCube` changes. If the set grows by one, at least one new view is generated. In other words, when a new component is created, the view having the component as *subject* will automatically be available. The reverse case when the set shrinks by one is analogous.

The cells to which the views are assigned describe the content of the views. Hence we can say that the view content of a view depends on the dynamic dimension values which define the cells. Figure 6 shows a view which is assigned to a cell. The view has a language part because of the assignment to a `SubCube`, and it has a content part because of the automatic assignment to a `Cell`.

View Projection using Cells

This section extends the view projection definitions from Sect. 5.2.2 using the definitions from the previous sections concerning the hyper-cube. We can now replace the `Environment` element by the hyper-cube definitions, since the hyper-cube

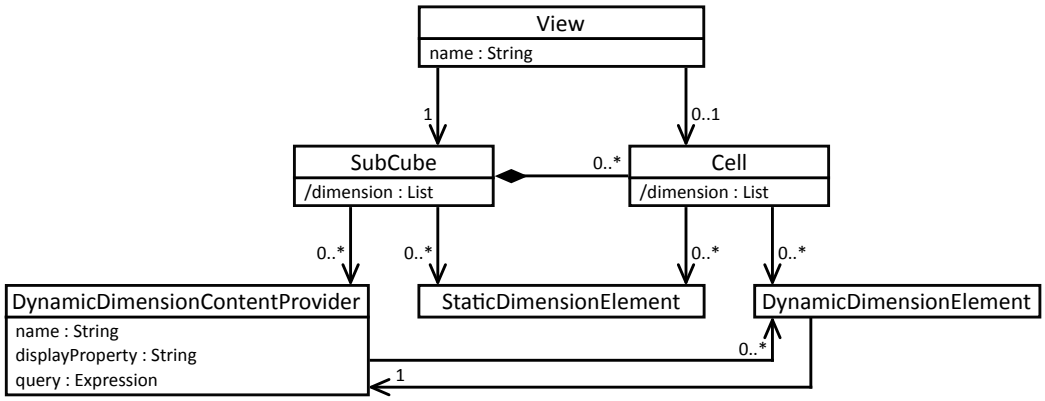

Figure 17: The Concept of a Concrete Cell

provides the environment for views using dimensions and cells. The updated structure is shown in Fig. 18.

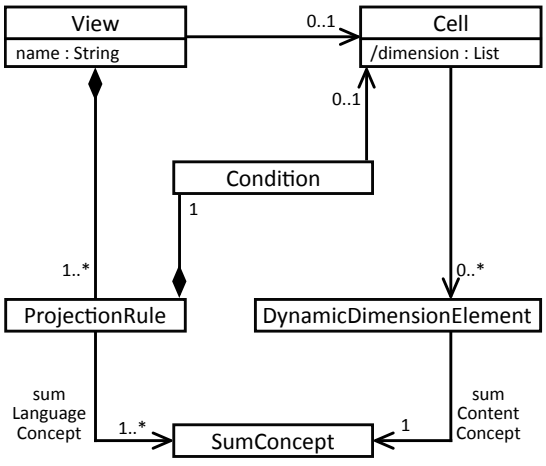

Figure 18: View Projection-Rule and Cell Properties

The projection rules of a view, project content from the SUM using concepts from the SUM language, which are accessable via the `sumLanguageConcept` relation. In the projection rule, all instances of the `sumLanguageConcept` are retrieved by "`allInstances()`"-like operations and projected to the content part of the view. The `Condition` of a `ProjectionRule` serves as a filter for the projection rule. The `Condition` influences the `ProjectionRule` using the information about the elements to which the `View` is related. The parameters of the environment are now the `dynamicDimensionElements` of the `Cell`. Since the `DynamicDimensionElements` are derived from the SUM content, they hold the corresponding information in the `sumContentConcept` relation. A projection rule used to project components from the SUM to a view can be controlled by a condition which uses the dynamic dimension value of the component dimension to project only the component which is assigned to the dynamic dimension value. The condition can also be used to query any possible content from the SUM, using query languages like the OCL (OMG 2011c), ModelJoin (Burger et al. 2016) or other mechanisms like decision trees (Breiman 1984).

6 Deep Orthographic Modeling Example

To show how the deep modeling approach described in the previous section would be applied in practice, and demonstrate that it at least has the capabilities of exiting EA modeling frameworks, in this section we apply it to the MEMO example presented in Sect. 3. More specifically, we show how our framework can be configured to support a part of the MEMO Framework based on that example. We therefore extended the prototype implementation described by Atkinson et al. (2013a) to support the approach described in the previous section. The ability to configure the orthographic modeling environment to a particular view-based method with support for deep modeling and dimension-based view navigation, significantly extends the power of the tool. The

current tooling implementation, which we call DOREEN, uses the ECLIPSE-based MELANEE tool as the underlying deep modeling platform and the DEEP-ATL transformation language to realize the deep modeling projections.

In this case study, we focus on the behavioral part of the enterprise specification – the other parts can be realized in a similar way. The behavioral part of an enterprise is mainly specified by the *aspect* dimension's "process" value (cf. Fig. 3). The values of the *perspective* dimension (i. e. "strategy", "organisation" and "information system") are used to specify the enterprise's behavior at different abstraction levels. These range from strategic views (e. g. for defining value chains) to implementation views (e. g. for defining the underlying information systems).

At the strategical level, the value chains of the enterprise are described. In the MEMO Framework, value chains are composed of the activities to be performed at the strategical level. These are further used in the description of business processes, situated at the organizational level. The mapping of the activities to business processes shows how the activities are realized by the underlying business. In this example we focus on this part of the MEMO Framework and realize it using two views, one view at the strategical level and one at the organizational level. The SUM is tailored to this small excerpt of the MEMO Framework and provides the needed concepts to capture the information conveyed by the two identified views.

The method configuration for this example is intentionally incomplete since it is only intended to provide an idea of how the MEMO approach can be supported in the orthographic modeling framework. Since the artifacts for the models (i. e. the SUM and the views) are represented using deep modeling technology (Gerbig 2017), the models can be easily extended to capture other parts of the MEMO Framework. The extension of the orthographic modeling environment is performed by a person playing the role of the methodologist, and requires the addition of further model elements to the SUM language and further views to support all of the *perspective* and *aspect* values. Since the extension would include the modification of the deep modeling language artifacts, a redeployment of the environment based on the changes is not needed. This is because the meta-model of the deep models can be manipulated at run-time with the help of emendation services to maintain the overall consistency of a deep model (Atkinson et al. 2012). The deep models for the example are shown using the general purpose visualization, but can be easily adapted to a MEMO domain specific visualization by creating the appropriate visualizers.

6.1 SUM Language

In order to provide a single consistent description of the enterprise under discussion, all information from the available views must be integrated into a SUM which provides a detailed and full specification of the enterprise. The SUM language for the part of the MEMO Framework relevant to this case study is shown in Fig. 19. The shown SUM language is a simplified version of the more detailed meta-model presented by Frank (2014). The language comprises concepts to capture the activities of a value chain using the `Activity` model element. The realization of the activities on the organizational level is represented by the `implementedBy` model element, linking the activity to one or more `BusinessProcesses`. The business processes are further described using the model elements `Process` and `Event`. The transition between the `Processes` to the `Events` and vice versa is realized by the `linkPE` model element. The deep model in the figure is presented using the general purpose visualization. The entities are presented using the rectangular notation, while the connections are presented using the collapsed notation, which renders a connection as a line with a black dot.

The model elements used to define the SUM language are enhanced with `potency` and `durability` properties to define the influence range of the model elements. Since `BusinessProcesses` can be executed in running enterprises, the `BusinessProcess` model element has a `potency`

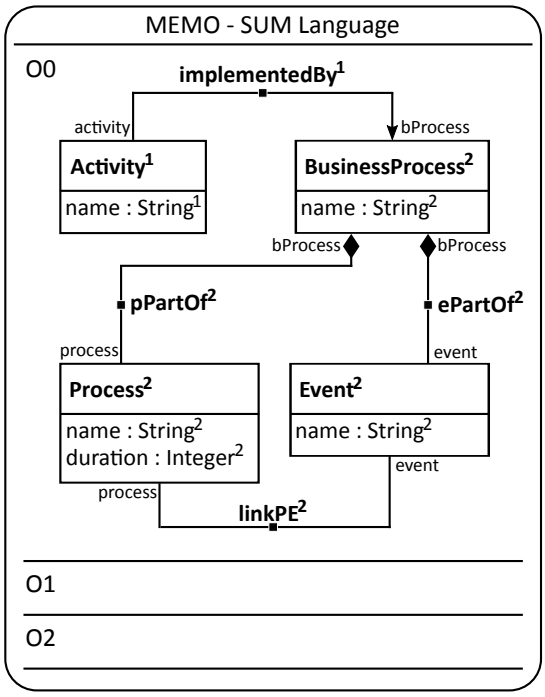

Figure 19: MEMO – SUM Language

value of "2" so that it can have instances at ontological level O_1 and O_2. The same is true of the model elements belonging to `BusinessProcess` such as (`Process` and `Event`) and the corresponding connections (`pPartOf`, `ePartOf` and `linkPE`). The attributes of these model elements are also defined with the same influence range as their owning model elements. This means it is possible to use the attributes in the ontological levels O_1 and O_2. The `duration` attribute of the model element `Process` is used in the ontological level O_2 to store the execution duration of a `Process` in the running enterprise.

6.2 View Language

There are two views of relevance for this example. One, the activities view, provides the ability to create activities (instances of `Activity`) at the strategic level and one or many business processes (instances of `BusinessProcess`) used to realize the activities. The other, the business process view, is used to further describe the business processes by defining the actions and events which occur within them. Since the second view is more interesting, we will focus on this view here. Figure 20 shows the view language for the business process view. The `BusinessProcess` model element in this view is the *subject* of the view since it is meant to be the view *of* a particular business process. The *subject*-oriented approach for defining views avoids the creation of views that convey information about an arbitrary part of the enterprise and enhances the structure of the enterprise specification.

The deep model used for the view language of the business process view as shown in Fig. 20 is able to capture types of `BusinessProcesses` in the ontological level O_1 as well as their instances situated in O_2. The `BusinessProcesses` in O_2 represent the executed business processes in the running enterprise. While most of the defined model elements can be used at both the O_1 and O_2 ontological levels, this is not the case for the `avgDuration` attribute of the `BusinessProcess` model element. This attribute is only available until O_1. The value of this attribute is derived dynamically in the view projection step and aggregated to show the average duration of a `BusinessProcess` type, using the duration information captured in the instances of the `BusinessProcess`. Since this attribute depends on the subsequent ontological level, it does not make sense to include it in O_2 since there are no `BusinessProcesses` at O_3. This ability to derive information from across many levels of abstraction is also supported in the MEMO Framework by applying the stereotype "«obtainable»" to an attribute. In the context of deep modeling, this concept can be applied over many abstraction levels (e. g. to derive some value in O_1 from O_3).

The second derived attribute in the presented view language is the `duration` attribute of `BusinessProcess`. This attribute shows the time needed for a `BusinessProcess` instance (on O_2) to finish by summing up the `duration` values of the `Processes` belonging to that `BusinessProcess`.

In the current example, most of the model elements used to define the view language are similar to the corresponding model elements in the SUM language, but this is not always the case. The presented framework allows almost any

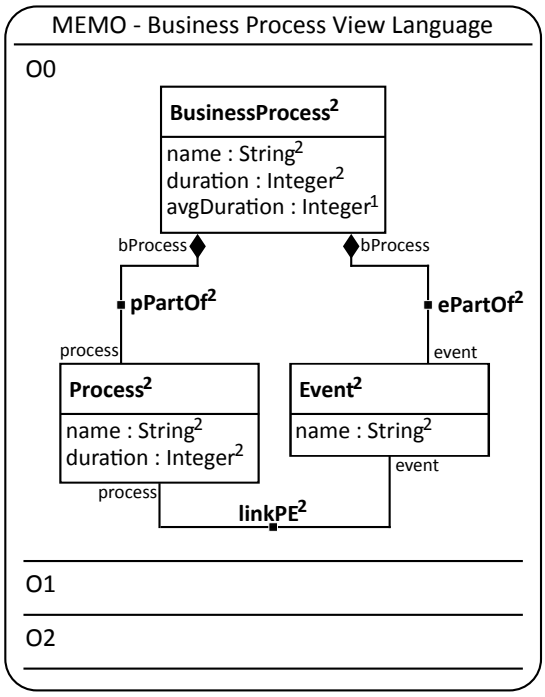

Figure 20: MEMO – View Language of Business Process View

arbitrary mapping between the model elements from the SUM and their counterparts in the views. This mapping can be defined in the projection rules based on the goal and intention of a view. The complexity of the mapping influences the degree of back propagation of the view's information to the SUM when a `BusinessProcess` is changed at the O_1 level for example. In pure read-only views, which are mostly used for reporting, the complexity of the projection rules does not matter since no back propagation is intended.

6.3 View Projection

In order to project information from the SUM to the views and propagate the information from the views back to the SUM, view projections are created as described in Sect. 5.2.2. In this example, ATL (ATLAS Transformation Language) (Bézivin et al. 2003) is used to query the appropriate concepts from the SUM and project them to the view. However, the approach is not strictly bound to ATL, other transformation languages like QVT (OMG 2011b) can also be used. Listing 1 shows the ATL code representing the view projection part of the view, shown in Fig. 20. The ATL code is written in the DEEP-ATL dialect which can be used to transform deep models to deep models, deep models to classical two-level models or classical two-level models to deep models. The dialect extends the standard ATL language with features to facilitate the direct selection of ontological concepts and their ontological properties. A detailed description of DEEP-ATL is available in Atkinson et al. (2013b).

In order to support the projection of multiple ontological levels from a source deep model into a target deep model, we extended the DEEP-ATL dialect with the concept of "rule potency". The rule potency of the source pattern defines the range of ontological levels which should be transformed. The ranges are defined by specifying a start and an end ontological level relative to the ontological level of the concept used in the pattern. For instance, the source pattern in Listing 1 line 5, selects all instances of the $O_0.BusinessProcess$ ontological concept situated at the ontological levels O_1 and O_2. The rule potency of the target pattern has the same semantics – it defines which ontological level the selected concepts should be added to. An example of the rule potency for a target pattern is contained in line 7 of the example ATL code. Since the value is equal to the rule potency of the source pattern, the rule transforms the instances (from O_1) of the $O_0.BusinessProcess$ ontological concept and the instances of the instances (from O_2), to instances (to O_1) of the target pattern concept $O_0.BusinessProcess$ and to instances of the instances (to O_2). If for example, only the business process instances from the ontological level O_2 from the SUM are needed in a view, then the rule potency value of "2" must be applied to the source pattern. This functionality allows complete classification hierarchies to be easily transformed from one deep model to another deep model.

```
1 -- @atlcompiler atlMLMcompiler
2
3 rule BusinessProcess2BusinessProcess {
4   from
5     s : SUM!O0.BusinessProcess 1..2 (
          thisModule.isSubject(s))
6   to
```

```
7       t : VIEW!O0.BusinessProcess 1..2 (
8         name <- s.name,
9         _l_.name <- s._l_.name,
10        duration <- thisModule.
              sumDurationOfProcesses(s),
11        avgDuration <- thisModule.
              avgDurationOfInstances(s)
12    )
13  }
14  rule Process2Process {
15    from
16      s : SUM!O0.Process 1..2 (thisModule.
            processBelongToSubject(s))
17    to
18      t : VIEW!O0.Process 1..2 (
19        name <- s.name,
20        _l_.name <- s._l_.name,
21        duration <- s.duration
22    )
23  }
24  rule pPartOf2pPartOf {
25    from
26      s : SUM!O0.pPartOf 1..2 (thisModule.
            pPartOfBelongToSubject(s))
27    to
28      t : VIEW!O0.pPartOf 1..2 (
29        bProcess <- s.bProcess,
30        process <- s.process
31    )
32  }
33  rule Event2Event {
34    from
35      s : SUM!O0.Event 1..2 (thisModule.
            eventBelongToSubject(s))
36    to
37      t : VIEW!O0.Event 1..2 (
38        name <- s.name,
39        _l_.name <- s._l_.name
40    )
41  }
42  rule ePartOf2ePartOf {
43    from
44      s : SUM!O0.ePartOf 1..2 (thisModule.
            ePartOfBelongToSubject(s))
45    to
46      t : VIEW!O0.ePartOf 1..2 (
47        bProcess <- s.bProcess,
48        event <- s.event
49    )
50  }
51  rule linkPE2linkPE {
52    from
53      s : SUM!O0.linkPE 1..2 (thisModule.
            linkPEBelongToSubject(s))
54    to
55      t : VIEW!O0.linkPE 1..2 (
56        process <- s.process,
57        event <- s.event
58    )
59  }
```

Listing 1: ATL Transformation for MEMO Business Process Views

The view projection definition contains six ATL `rules`, each responsible for the projection of one concept. The projection is defined on the type-level (i. e. in terms of the concepts contained in the view and SUM language). In this case the concepts are contained in the O_0 ontological level of the deep models. The rule *BusinessProcess2BusinessProcess* projects ontological instances of the O_0.*BusinessProcess* concepts from the SUM to the view as ontological instances of the O_0.*BusinessProcess* ontological concept, defined in the view language. Within the ATL rules, the properties of the concepts can also be projected, e. g. the linguistic and ontological *name* properties (line 8 and 9) of the O_0.*BusinessProcess* concepts. Here the *_l_.name* (line 9) corresponds to the linguistic *name*, whereas the *name* (or *_o_.name*) corresponds to the ontological *name* property. The shorthand notation for the ontological properties can be used since the concepts in the `from` and `to` part of the ATL rule are ontological concepts.

While the concepts of the *BusinessProcess2BusinessProcess* rule are linguistic `Entities` their properties are linguistic `Attributes`. The rule *linkPE2linkPE* on the other hand projects concepts which are linguistic `Connections`. So the binding of the properties defined in the rule includes linguistic `Attributes` and `ConnectionEnds`. The latter are used to navigate from a `Connection` to an `Entity`. In the rule *linkPE2linkPE* in line 56 and 57 the `ConnectionEnd` properties *process* and *event* are used.

Most of the concepts' properties are directly projected to their counterparts (e. g. the *name* properties). This is because the view language largely overlaps with part of the SUM language. The property assignments in the lines 10 and 11 are an exception. As defined in the view language the property *duration* shows the duration of the execution of a business process instance (on O_2). Therefore the helper *sumDurationOfProcesses()* summarizes the duration property values of all process instances (on O_2), belonging to the given business process and returns the result as

an `Integer`. The property *avgDuration* shows the average duration of all business process instances. To this end, the helper *avgDurationOfInstances()* computes the average of the duration property values over all business process instances. This aggregation of information crosses the type/instance boundary since information from O_2 is aggregated and stored in O_1.

As mentioned earlier, the view showed in the example overlaps significantly with the SUM, but this need not be the case. In general, our presented approach allows any kind of view to be projected from the SUM. The derivation can be performed directly (e. g. line 8) or using some aggregation mechanisms (e. g. line 10). Furthermore, the aggregation is not limited to properties of concepts. It is possible to aggregate many concepts from the SUM and generate one single concept in a view, or to generate many concepts in a view based on one single concept from the SUM. Since the prototype relies on ATL, all capabilities of the transformation language (helpers, do-blocks, lazy rules,. . .) can be used to derive the contents of views. By aggregating information, it is possible to create views which act like reporting views in business intelligence. For instance, it is possible to show the frequency of the execution of a particular `Process` by considering all executions of a `BusinessProcess` type. Another example from the software modeling area is to create a view which shows a `Class` enhanced by the concepts which it inherits based on its inheritance tree.

Each rule in Listing 1 has a filter which in ATL is placed in the `from` part of a rule. Note that the ATL helpers used by the filters are omitted in the ATL module shown in order to enhance readability. The role of a filter is to ensure that only the needed concepts from the SUM are projected to the view using information from the orthographic modeling environment (i. e. what/who is the *subject* of the view and how are the other concepts related to the *subject*). The views in orthographic modeling should be minimal in terms of the number of concepts they support, but they should be sufficiently expressive to fulfill their purpose (Atkinson and Tunjic 2014a). All the filters in the shown ATL module depend on the *subject* parameter which is provided by the orthographic modeling environment. The resulting view is thus a *subject*-oriented view since it omits contain concepts which are not relevant from the perspective of the view's *subject*.

6.4 Hyper-Cube

In order to provide a navigation mechanism for the MEMO Framework a hyper-cube must be defined in terms of orthogonal dimensions as described in Sect. 5.3. Each cell in the resulting hyper-cube is identified by a collection of dimension's values (i. e. coordinates) and may or may not be associated with a concrete view. In the following, we first show the hyper-cube configuration used in our example and then show the configuration applied to an exemplary SUM to obtain concrete dynamic dimension elements using content providers.

The MEMO high-level Framework is based on two orthogonal dimensions – *aspects* and *perspectives*. In this example, we use these two dimensions and their values as static dimensions for the hyper-cube. We further define a dynamic dimension with the name *BusinessProcesses* which is used to show all `BusinessProcess` model elements contained in the SUM. It is of course possible to define further dynamic dimensions for further views, but to keep the example as simple as possible we use only one dynamic dimension to demonstrate the capabilities of our approach. We define therefore the `DynamicDimension` *BusinessProcesses*, which contains a `DynamicDimensionContentProvider`. This content provider provides the `DynamicDimensionElements` for this dynamic dimension. Since the underlying deep modeling platform is able to capture information which span many abstraction levels, and the example SUM spans three ontological levels ($O_0 \ldots O_2$), we use the nesting dimension concept to represent instances of the business process types. We also define a second `DynamicDimensionContentProvider` which is contained by the first via a `child` relation. Figure 21 shows the configured hyper-cube

for the MEMO-based example. Again, to enhance readability the following abbreviations are used to refer to the dimension elements: `StaticDimension` (SD), `DynamicDimension` (DD), `StaticDimensionElement` (SDE) and `DynamicDimensionContentProvider` (DDCP).

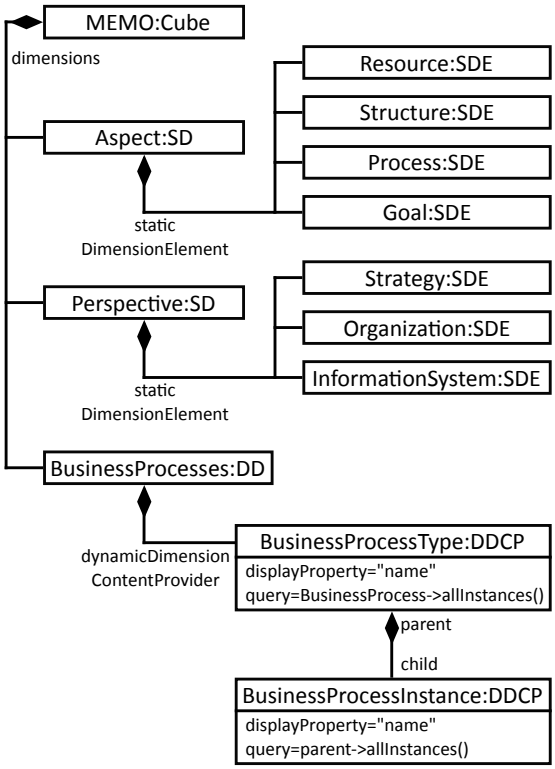

Figure 21: MEMO – Hyper-Cube

Using the project configuration information, the content providers for the dynamic dimension extract the appropriate dynamic dimension elements so they can be shown as values of the dynamic dimension. Figure 22 shows the application of the shown configuration to the MEMO example. The three tables (Aspects, Perspectives and BusinessProcesses) represent the dimensions, while their contents represent the dimension values. A cell is selected when a dimension value for each dimension is picked. In Fig. 22 the selected cell has the values (i. e. coordinates) "*Process – Organization – CPC_exec-01*".

The SUM in the example has two instances of business process on the ontological level O_1

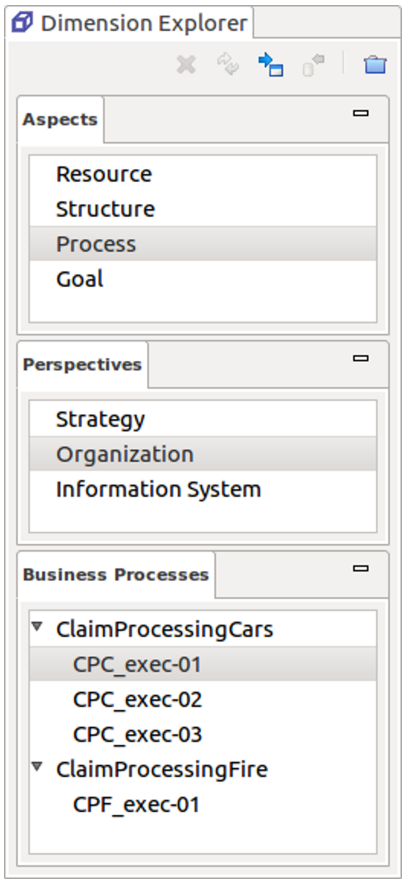

Figure 22: MEMO – Hyper-Cube with Navigation Dimensions

(*ClaimProcessingCars* and *ClaimProcessingFire*). These are queried by the *BusinessProcessType* content provider using the query "BusinessProcess->allInstances()". The operation *allInstances()* returns all instances of a model element situated at the next ontological level. The two instances of business processes in the SUM are further types of their instances which are situated on the ontological level O_2 (*CPC_exec-01*, *CPC_exec-02*, *CPC_exec-03* and *CPF_exec-01*). These are queried by the nested content provider *BusinessProcessInstance* which calls the operation *allInstances()* on the business process types provided by their content provider. Finally, the instances of the business processes (from O_1 and O_2) are displayed as `DynamicDimensionElements` in the `DynamicDimension` *BusinessProcesses* us-

ing a tree structure derived from the structure of the used content providers. This shows business process instances as children of the corresponding business process types.

The presented approach provides the mechanisms to define any needed cell using the concepts of static and dynamic dimensions. For the current MEMO example, it would also be possible to show the sub-processes of the business process types in the nesting dimension and show the business process instances in a further dynamic dimension (e. g. *BusinessProcessInstances*). In this case the relationship between the dynamic dimension elements of the *BusinessProcess* dimension would be of kind "contained-by", rather than "instance-of", since the model elements in the top dimension would be the containers of the model elements in the nested dimension. This configuration would be equivalent to OLAP drill-down/roll-up operations. Beyond the shown use cases the hyper-cube can also be used to create views which are used for reporting, as in business intelligence. In this case the dynamic dimensions could be used as the grouping and filtering criterion which let the architect dynamically control the content of views. The interpretation of the dynamic dimension elements used for grouping and filtering would be realized using aggregations in the projection rules. Technically these would be handled by the underlying transformation language.

6.5 Views in the Hyper-Cube

As described in Sect. 5.3.3, views are assigned to cells of the hyper-cube using so-called `SubCubes`. The `SubCubes` are sub-cubes of the hyper-cube which are defined by `StaticDimensionElements` and `DynamicDimensionContentProviders`. The business process view in the example is assigned to the `SubCube` which is defined by the *Organization* and *Process* `StaticDimensionElements` and the `DynamicDimensionContentProvider` which provides all business processes available in the SUM.

Figure 23 shows the `SubCube` of the *BusinessProcessView* from Fig. 20, based on the hyper-cube configuration shown in Fig. 21.

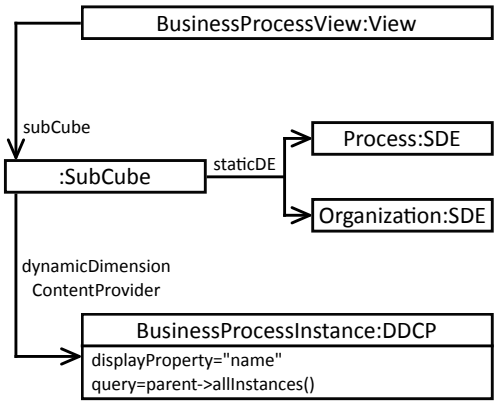

Figure 23: MEMO – SubCube of Business Process View

The `SubCube` is defined by the *Process* and *Organization* `StaticDimensionElements` (`SDE`) and the *BusinessProcessInstance* `DynamicDimensionContentProvider` (`DDCP`). The *BusinessProcessInstance* `DynamicDimension-ContentProvider` is used to return the instances of business process types. For this purpose it uses the business process from its parent `DynamicDimensionContentProvider` as can be seen from the `query` expression. Note that the view content part of the view shown in Fig. 23 is empty since no concrete value for the dynamic dimension is available at this point.

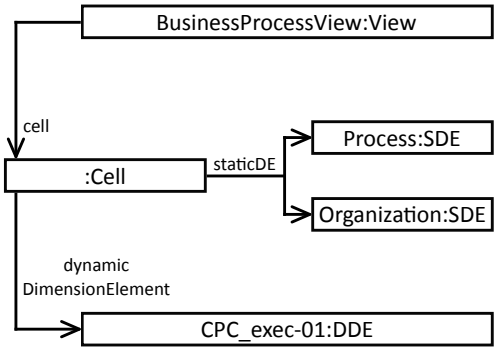

Figure 24: MEMO – Cell of Business Process View

Since the orthographic modeling environment uses the method configuration in a project, the `SubCubes` are used to generate the `Cells` which ultimately contain the view content. Based on the Figures 22 and 23, Fig. 24 shows the generated `Cell`

for the *CPC_exec-01* business process instance. The `DynamicDimensionContentProvider`'s `DynamicDimensionElement` (`DDE`) with the name "CPC_exec-01" has been picked, leading to a concrete cell which is defined by a single dimension value from each dimension. Besides the "CPC_exec-01" cell, further cells are generated for all dynamic dimension elements returned by the content providers. The content of the view, shown in Fig. 24, is not empty since it contains at least the *subject* of the view – the *CPC_exec-01* business process instance. As well as the business process instance, the view can contain further concepts which are relevant for the *subject*, e. g. instances of the sub-processes, events and relations.

Finally, Fig. 25 shows two versions of the concrete content of the view which is contained in the selected cell shown in Fig. 22. On the left-hand-side of the double dashed vertical line (a), the view content is shown using the default general purpose visualization and on the right-hand-side (b) it is shown using a domain specific visualization. While the first ontological level (O_0) contains the view language, the content is situated in the second and third ontological level. In this example, the ontological level O_1 contains the architectural information and the O_2 the operational information. The concepts in O_1 and O_2 are projections of the corresponding concepts from the SUM created according to the defined projection rules. The correspondence between the view and the SUM concepts is further captured by the projection traces which are generated by the projection rules in the projection process. The concepts in the view content are ontological instances of the concepts from the view language. The view content of the presented view spans two ontological levels since it shows a business process instance along with its type. According to the hyper-cube, the instance "CPC_exec-01" is picked as the *subject* for the view. Based on the projection rules from Listing 1 both the business process instance and its business process type are projected into the view. This is defined by the rule potency value in the ATL rules.

In order to realize the domain specific visualization shown in Fig. 25 (b) every concept from the view language (see Fig. 20) needs to be provided with a corresponding visualizer. Since the instances (O_1 and O_2) "inherit" the visualizers from the types (O_0), the concepts contained in the view content part will automatically be rendered using the domain specific visualizers. The domain specific visualizers are taken from the original example from Frank (2002) but are slightly modified to fit to our example. The modification basically ignores the distinction between "manually", "semi-automatically" or "automatically" executed processes. In order to support the different types the SUM language, view language and view projection rules need to be extended. In the view language the three types would be modeled using one concept per process type, whereas in the SUM they could be mapped to a single concept. The concept's type in the SUM can be identified using an attribute. This is an example of the "information expansion" approach, described in in Sect. 5.1. The view presented in Fig. 25 shows the two properties obtained by aggregating information about operations executed in the projection process. The *duration* property of the *CPC_exec-01* model element contains the sum of all duration values of the owned `Process` instances in milliseconds. The domain specific visualization provides a more meaningful way of presenting the property – the value is converted from milliseconds to seconds with the corresponding unit indicator as suffix. The second aggregated value in this view is the *avgDuration* of the *ClaimProcessingCars* model element. This property value is derived by accumulating all *duration* property values of all instances of the *ClaimProcessingCars* model element (in O_2). Due to the `durability` property value ("1") of the *avgDuration* `Attribute` (see Fig. 20), the property only exists at the O_1 ontological level and not at the following levels. This value indicates that the execution duration of the *CPC_exec-01* business process instance is above the average execution duration of over all business process instances.

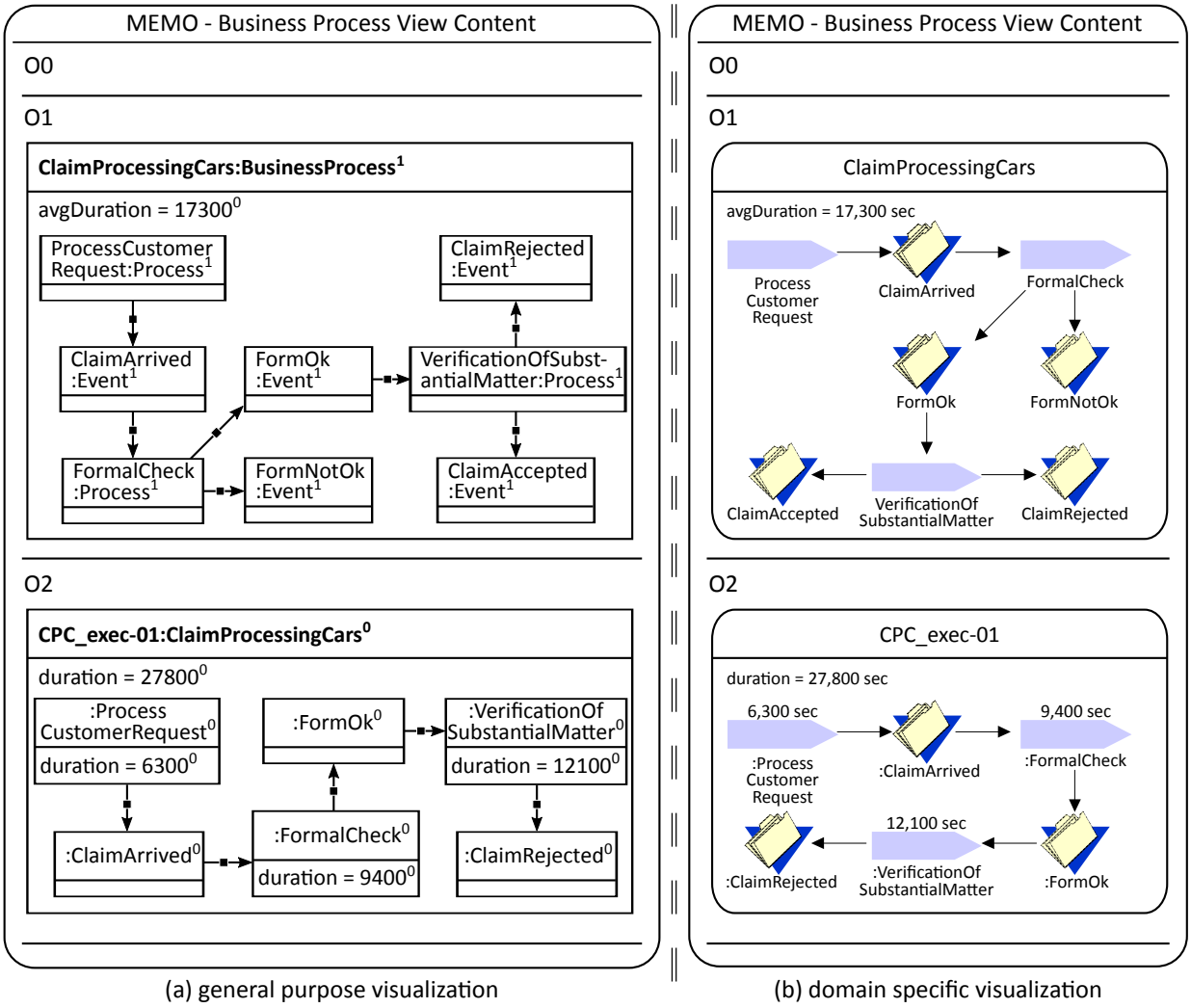

Figure 25: MEMO – View Content of Business Process View with Subject CPC_exec-01 (Frank 2002)

7 Discussion

In this paper we have presented an approach for supporting Model Driven Organizations that leverages deep modeling and orthographic modeling in a unified, view-based environment to seamlessly support architectural and operational views of the underlying organization. In this section, we discuss how the presented approach realizes the goals set out in Sect. 4, and what benefits this offers to enterprises. We also discuss limitations and weaknesses.

7.1 Deep, Projective, Component-based, Composite Views

The key requirement for the approach, as outlined in Sect. 4, is a viewpoint framework that cleanly and fundamentally supports projective, component-based, composite views of an organization. The presented realization approach not only achieves all these goals, it does so in a generic way that can be easily customized and used by normal architects and administrators.

Projective views are supported by the fundamental principle of using a SUM to capture all knowledge about the organization under description and to generate views of the SUM, on demand, by the application of explicitly modeled transformations as shown in Listing 1. The languages for representing the view and SUM content (Figures 19 and 20), as well as the transformations for projecting information between them (Listing 1), are written using mature model-driven development principles (i. e. class-based structural modeling, ATL-like transformation definition etc.). New views can thus be added to the modeling environment using the basic skills of model-driven development.

Deep views are supported by realizing both, the SUM and the views using a multi-level modeling platform rather than a standard two-level platform. In our case we used the deep, multi-level platform known as MELANEE (Gerbig 2017). This provides two important capabilities for modeling views that are not available in traditional EA modeling approaches and tools. First, views can themselves be multi-level in that they represent information that exist at two or more levels of classification. This is illustrated by Fig. 25 which shows a process (type) definition (ClaimProcessingCars), at the O_1 level and an executed instance of that process (CPC_exec-01) at the operational O_2 level. Second, and more importantly, it allows views to represent information at any level (operation, type, meta-type,. . .), using the same modeling techniques.

Component-based views are supported by parameterizing all projection transformations by the model element in the SUM representing the component (i. e. part) of the organization that is being looking at. Thus, the various ATL rules in Listing 1 which are used to project business process views are explicitly parameterized by the business process instance chosen as the subject of a view. Figure 25 shows a view that results when one specific instance of one specific business process of the Insurance Sales organization is selected as the subject of the view. In the example viewpoint framework, developed in Sect. 6, views containing more than one business process cannot be created, because they cannot even be identified in the example hyper-cube, although this could of course be changed if desired by extending the hyper-cube and adding further views.

Finally, composite views, in which abstract views and concrete views can be nested, are supported by the dimension based navigation/identification system which uniquely integrates static, architectural dimensions and dynamic, operational dimensions. This is perhaps the most innovative aspect of the approach presented in this paper. Concrete views, which are intended to be physically rendered on some device or medium, must have all of their coordinates explicitly selected. Thus, the view shown in Fig. 25 corresponds to the unique set of coordinates shown in Fig. 22. On the other hand, abstract views, which are intended to encapsulate other abstract and concrete views, are not required to have all coordinates explicitly selected. They therefore correspond to sub-hyper-cubes or slices of the overall hyper-cubes traced out by the dimension space. Thus,

for example, the slice represented schematically in Fig. 16 by selecting the structural dimensions and leaving the rest unspecified is an abstract view providing a structural perceptive on the system. To actually see any information a concrete view (i. e. a sub-view of the abstract view) would have to be selected by specifying all the coordinates. The nesting of abstract and concrete views within abstract views therefore takes place by the natural OLAP-like metaphor of "slicing and dicing".

7.2 Model-Driven DevOps

The original motivation for the approach was to support the MDO vision by making it possible for all stakeholders in an organization to fulfill their assignments using representations of (parts of) an organization that best suit their skills and tasks. This includes stakeholders that are more interested in relatively static, architectural views such as developers and architects, and stakeholders who are more interested in dynamic and historical data, such as managers and system administrators. In essence, therefore, the key challenge was to support a model-driven approach to DevOps, in which the advantages of model-driven development could be seamlessly and uniformly exploited for both development and operation.

The approach presented in this paper achieves this goal by leveraging the synergy between the emerging paradigms of multi-level modeling and orthographic modeling. The first of these, multi-level modeling, provides the fundamental basis for the seamless integration of development and run-time views by allowing information across all levels of classification to be manipulated and represented using the same powerful feature of model-driven development. This is demonstrated explicitly by Fig. 25 in the example which shows operational (instance) information presented next to architectural (type) information within the same view using the same object-oriented principles of typing and language definition/application. Instance-level objects do not always have to be shown along with their types – views can also focus on just one ontological level of abstraction such as the instance-level or type-level or meta-type-level etc.

Orthographic modeling also plays a critical role in supporting the ability to work seamlessly with development and operational views at the same time because it is the key to making the views accessible within a single, unified viewpoint framework. More specifically, it provides the basis for integrating architectural concerns such as abstraction level, opaqueness and perspective (e. g. structural, behavioral, etc.) with operational concerns such as aggregation and instance analysis. As illustrated by Fig. 16, therefore, the set of coordinates used to identify a concrete view combines relatively static choices about what type of view is desired (i. e. structural, platform independent) with more dynamic choices about what the subject is (i. e. what the view is looking at).

Most EAM tools today cannot support the same level of flexibility and seamlessness. In most tools, view types are usually defined by programming a new kind of editor or dashboard using a standard programming language and relative low-level representation of data. The resulting views are therefore usually not created by model-based techniques and thus cannot benefit from the benefits of type safety or the productivity enhancements through domain specific languages. Moreover, when model-based techniques are used, the resulting views can only display information at one classification level immediately below the defined language due to the use of two-level modeling technology.

The other big problem with most EAM tools today is that the views are organized in relatively simple and ad-hoc ways in the form of some kind of tree with arbitrary nesting and naming conventions. Moreover, if they make some kind of distinction between the underlying model elements and views, they usually allow users to manipulate the underlying model elements directly and in arbitrary ways. The approach presented in this paper essentially balances the discipline and productivity advantages of model-driven engineering at development-time with the flexibility and convenience of OLAP-like data analysis at

operation-time. It therefore provides a realization of the interactive data warehouse vision presented by Draheim (2013).

7.3 Challenges and Future Work

Production-ready implementations of deep, orthographic modeling environments that can be used in industrial projects are clearly still a long way off, and their development presents numerous challenges. First, the dynamic, on-the-fly generation and updating of views whenever the SUM is changed presents some problems for rendering them, especially in graphical forms. Users of graphical models are usually frustrated when the layout of models changes in between re-renderings. Finding a good solution to this problem is an open research question, but should be addressable by retaining partial layout information at the client side (i. e. the computers used to visualize views).

There are a lot of issues related to the initial creation and evolution of SUMs. For example, given that most companies have many legacy metamodels and tools, how can they be integrated into a SUM and how can the SUM be changed? These are challenging questions, but no harder than the challenges involved in maintaining multiple data source. Other research teams are working on this specific problem (Burger et al. 2016).

Finally, like all new paradigms there are a lot of issues related to the usability and uptake of the approach. It is not only unclear what viewpoint frameworks (i. e. dimension spaces) and view types are best suited for particular domains, it is a major challenge to migrate to the new approach and help stakeholders learn how to use it effectively.

8 Conclusion

The core idea behind the MDO vision is to allow all stakeholders in an organization to fulfill their assignments using representations of (parts of) that organization that best suit their skills and tasks. This includes stakeholders who are mainly interested in static, architectural aspects of an organization such as architects and methodologists, and stakeholders who are more interested in dynamic or historical aspects of a system such as line managers and administrators. The contributions of this paper are (a) to make the case that a deep, orthographic modeling framework provides the best platform for realizing such a vision, based on experiences in building/using enterprise systems (Draheim 2010; Draheim and Weber 2002, 2003a,b) and on an analysis of the state-of-the-art in EA modeling, and (b) to demonstrate the feasibility of the approach by means of a prototype realization.

Three key innovations were needed to develop a viewpoint framework to achieve this goal. The first was to find a way of mixing static, architectural concerns and dynamic, operational concerns into a single, dimension-based paradigm for navigating around concrete and abstract views and allowing them to be nested. The second was to find a way of defining transformations (i. e. projections) that could (a) be parameterized by the subjects of views as well as their types and (b) allow view subjects to be derived at run-time from the SUM via the dimension-based navigation scheme. The third was to find a way of generalizing the two previous capabilities so that they support and leverage deep SUMs, views and transformations, and allow viewpoint frameworks to be configured by methodologists for different methods (i. e. constellations of views).

While the presented realization and accompanying prototype demonstrates the basic feasibility of the approach, as explained in Sect. 7 there are numerous challenges still to be overcome and many questions still to be answered. Our research has essentially reached the “design artifact” phase of the design sciences research approach (Hevner et al. 2004; Offermann et al. 2009). The next step is to systematically generate and evaluate experimental data as well as quantitative and subjective feedback on the MDO approach. In other words, we are about to enter the case-study/action research phase (Runeson et al. 2012; Sein et al. 2011) in which we will exercise the prototype on a realistic scenario.

References

Aßmann U., Götz S., Jézéquel J.-M., Morin B., Trapp M. (2014) A Reference Architecture and Roadmap for Models@run.time Systems In: Models@run.time: Foundations, Applications, and Roadmaps Bencomo N., France R., Cheng B. H. C., Aßmann U. (eds.) Springer, Cham, pp. 1–18

Atkinson C. (2002) Component-based Product Line Engineering with UML. Component Software Series. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA

Atkinson C., Draheim D. (2013) Cloud Aided-Software Engineering – Evolving Viable Software Systems through a Web of Views. In: Software Engineering Frameworks for the Cloud Computing Paradigm, pp. 255–281

Atkinson C., Gerbig R., Kennel B. (2012) On-the-fly emendation of multi-level models. In: European Conference on Modelling Foundations and Applications Lecture Notes in Computer Science 7349, pp. 194–209

Atkinson C., Gerbig R., Tunjic C. (2013a) A multi-level modeling environment for SUM-based software engineering. In: Proceedings of the 1st Workshop on View-Based, Aspect-Oriented and Orthographic Software Modelling VAO '13, 2:1–2:9

Atkinson C., Gerbig R., Tunjic C. V. (2013b) Enhancing classic transformation languages to support multi-level modeling. In: Software & Systems Modeling 14(2), pp. 645–666

Atkinson C., Kühne T. (2001) The Essence of Multilevel Metamodeling. In: Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools. Springer, London, UK, pp. 19–33

Atkinson C., Kühne T. (2002) Rearchitecting the UML Infrastructure. In: ACM Transactions on Modeling and Computer Simulation 12(4), pp. 290–321

Atkinson C., Stoll D., Bostan P. (2010) Orthographic Software Modeling: A Practical Approach to View-Based Development. In: International Conference on Evaluation of Novel Approaches to Software Engineering Communications in Computer and Information Science, pp. 206–219

Atkinson C., Stoll D., Tunjic C. (2011) Orthographic Service Modeling. In: 2011 IEEE 15th International Enterprise Distributed Object Computing Conference Workshops. IEEE, pp. 67–70

Atkinson C., Tunjic C. (2014a) Criteria for Orthographic Viewpoints. In: Proceedings of the 2nd Workshop on View-Based, Aspect-Oriented and Orthographic Software Modelling - VAO '14. ACM, New York, New York, USA, pp. 43–50

Atkinson C., Tunjic C. (2014b) Towards Orthographic Viewpoints for Enterprise Architecture Modeling. In: 2014 IEEE 18th International Enterprise Distributed Object Computing Conference Workshops and Demonstrations. IEEE, pp. 347–355

Atkinson C., Tunjic C. (2016) Towards a Configuration Framework for Orthographic-Software-Modeling Environments. In: 4th Workshop on View-Based, Aspect-Oriented and Orthographic Software Modelling - VAO '16 Karlsruhe Reports in Informatics (2016,7), pp. 7–10

Atkinson C., Tunjic C., Möller T. (2015) Fundamental Realization Strategies for Multi-View Specification Environments. In: 2015 IEEE 19th International Enterprise Distributed Object Computing Conference (EDOC), pp. 40–49

Belaunde M., Burt C., Casanave C., et al (2003) MDA Guide Version 1.0.1 Object Management Group (OMG) http://www.omg.org/news/meetings/workshops/UML_2003_Manual/00-2_MDA_Guide_v1.0.1.pdf Last Access: 2018-03-28

Bézivin J., Dupé G., Jouault F., Pitette G., Rougui J. E. (2003) First experiments with the ATL model transformation language: Transforming XSLT into

XQuery. In: 2nd OOPSLA Workshop on Generative Techniques in the context of Model Driven Architecture. Vol. 37, pp. 1–18

Bittmann S. (2014) Cooperative-Intrinsic Planning and Model-Driven Design of Business Information Systems. In: 44. Jahrestagung der Gesellschaft für Informatik (GI). Lecture Notes in Informatics Vol. 232, pp. 2281–2286

Bock A. (2015) Beyond Narrow Decision Models: Toward Integrative Models of Organizational Decision Processes. In: Proceedings of the 17th IEEE Conference on Business Informatics. IEEE Computer Society, Lisbon, Portugal, pp. 181–190

Bock A., Frank U. (2016) MEMO GoalML: A context-enriched modeling language to support reflective organizational goal planning and decision processes. In: Comyn-Wattiau I., Tanaka K., Song I., Yamamoto S., Saeki M. (eds.). Lecture Notes in Computer Science Vol. 9974. Springer, Cham, pp. 515–529

Brand S. (2015) Magic Quadrant for Enterprise Architecture Tools. G00271052. Gartner Inc

Breiman L. (1984) Classification and regression trees. The Wadsworth statistics/probability series. Wadsworth International Group, Belmont, Calif.

Budinsky F., Steinberg D., Merks E., Ellersick R., Grose T. J. (2003) Eclipse Modeling Framework: A Developer's Guide. Addison-Wesley, Boston, Mass.

Burger E., Henss J., Küster M., Kruse S., Happe L. (2016) View-based model-driven software development with ModelJoin. In: Software & Systems Modeling 15(2), pp. 473–496

Clark T., Frank U., Kulkarni V., Barn B. S., Turk D. (2013) Domain Specific Languages for the Model Driven Organization. In: First Workshop on the Globalization of Domain Specific Languages GlobalDSL '13, pp. 22–27

Clark T., Willans J. (2013) Software Language Engineering with XMF and XModeler. In: Formal and Practical Aspects of Domain-Specific Languages. IGI Global, USA, pp. 311–340

Codd E., Codd S., Salley C. (1993) Providing OLAP (On-line Analytical Processing) to User-analysts: An IT Mandate. Codd & Associates

Davis J., Daniels K. (2015) Effective Devops: Building a Culture of Collaboration, Affinity, and Tooling at Scale. O'Reilly Media

De Lara J., Guerra E., Cuadrado J. S. (2014) When and How to Use Multilevel Modelling. In: ACM Transactions on Software Engineering and Methodology 24(2), 12:1–12:46

Draheim D. (2010) The Service-Oriented Metaphor Deciphered. In: Journal of Computing Science and Engineering 4(4), pp. 253–275

Draheim D. (2012) Smart Business Process Management. In: 2011 BPM and Workflow Handbook, Digital Edition. Future Strategies, Workflow Management Coalition, pp. 207–223

Draheim D. (2013) Towards Total Budgeting and the Interactive Budget Warehouse. In: Innovation and Future of Enterprise Information Systems. Lecture Notes in Information Systems and Organisation, vol. 4, pp. 271–286

Draheim D., Weber G. (2002) Strongly Typed Server Pages. In: Halevy A., Gal A. (eds.) Next Generation Information Technologies and Systems. Springer, Berlin, Heidelberg, pp. 29–44

Draheim D., Weber G. (2003a) Modeling Submit/Response Style Systems with Form Charts and Dialogue Constraints. In: Meersman R., Tari Z. (eds.) On The Move to Meaningful Internet Systems 2003: OTM 2003 Workshops. Springer, Berlin, Heidelberg, pp. 267–278

Draheim D., Weber G. (2003b) Storyboarding form-based interfaces. In: Rauterberg G., Menozzi M., Wesson J. (eds.) Proceedings of INTERACT'03. IOS Press, pp. 343–350

Finkelstein A., Kramer J., Nuseibeh B., Finkelstein L., Goedicke M. (1992) Viewpoints: A Framework for Integrating Multiple Perspectives in System Development. In: International Journal of Software Engineering and Knowledge Engineering 2(1), pp. 31–57

Foster J. N., Greenwald M. B., Moore J. T., Pierce B. C., Schmitt A. (2007) Combinators for Bidirectional Tree Transformations: A Linguistic Approach to the View-update Problem. In: ACM Transactions on Programming Languages and Systems 29(3) (17)

Frank U. (1994) Multiperspektivische Unternehmensmodellierung: Theoretischer Hintergrund und Entwurf einer objektorientierten Entwicklungsumgebung. Oldenbourg Verlag

Frank U. (2002) Multi-perspective Enterprise Modeling (MEMO) – Conceptual Framework and Modeling Languages. In: Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS). IEEE, pp. 1258–1267

Frank U. (2011) MEMO Organisation Modelling Language (1): Focus on Organizational Structure. 48. Institute for Computer Science and Business Information Systems (ICB). University of Duisburg-Essen, Essen

Frank U. (2014) Multi-perspective enterprise modeling: Foundational concepts, prospects and future research challenges. In: Software & Systems Modeling 13(3), pp. 941–962

Frank U. (2016) Designing Models and Systems to Support IT Management: A Case for Multi-level Modeling. In: Proceedings of the 3rd International Workshop on Multi-Level Modelling. MULTI 2016, pp. 3–24

Gamma E., Helm R., Johnson R., Vlissides J. (1995) Design Patterns: Elements of Reusable Object-oriented Software. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA

Gerbig R. (2017) Deep, Seamless, Multi-format, Multi-notation Definition and Use of Domain-specific Languages English. PhD thesis, University of Mannheim

Hevner A. R., March S. T., Park J., Ram S. (2004) Design Science in Information Systems Research. In: MIS Quarterly 28(1), pp. 75–105

Iacob M. E., Jonkers H., Lankhorst M. M., Proper E., Quartel D. (2012) ArchiMate 2.0 Specification: The Open Group. Van Haren Publishing

IEEE Architecture Working Group (2000) IEEE Standard 1471-2000, Recommended practice for architectural description of software-intensive systems ANSI/IEEE-Std-1471-2000. IEEE

Igamberdiev M., Grossmann G., Selway M., Stumptner M. (2016) An integrated multi-level modeling approach for industrial-scale data interoperability. In: Software & Systems Modeling 17(1), pp. 269–294

Inmon W. H. (1992) Building the Data Warehouse. John Wiley & Sons, Inc., New York, NY, USA

ISO/IEC/IEEE (2011) Systems and Software Engineering – Architecture description ISO/IEC/IEEE 42010:2011. ISO

ISO/IEC/ITU-T (1997) RM-ODP. Reference Model for Open Distributed Processing ISO/IEC 10746, ITU-T Rec. X.901-X.904. ISO

Kennel B. (2012) A Unified Framework for Multi-Level Modeling. PhD thesis, University of Mannheim

Kimball R., Ross M. (2013) The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling. John Wiley & Sons, Inc., New York, NY, USA

Kirchner L. (2008) Eine Methode zur Unterstützung des IT-Managements im Rahmen der Unternehmensmodellierung. Logos-Verlag, Berlin, Germany

Kühne T. (2006) Matters of (meta-) modeling. In: Software & Systems Modeling 5(4), pp. 369–385

Lwakatare L. E., Kuvaja P., Oivo M. (2015) Dimensions of DevOps In: Agile Processes in Software Engineering and Extreme Programming: 16th International Conference, XP 2015, Helsinki, Finland Springer, Cham, pp. 212–217

Neumayr B., Schuetz C. G., Jeusfeld M. A., Schrefl M. (2016) Dual deep modeling: multi-level modeling with dual potencies and its formalization in F-Logic. In: Software & Systems Modeling 17(1), pp. 233–268

Offermann P., Levina O., Schönherr M., Bub U. (2009) Outline of a Design Science Research Process. In: Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology. DESRIST '09. ACM, New York, NY, USA, 7:1–7:11

OMG (2011a) Business Process Model and Notation (BPMN), Version 2.0 Object Management Group http://www.omg.org/spec/BPMN/2.0 Last Access: 2018-03-28

OMG (2011b) Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, Version 1.1 Object Management Group http://www.omg.org/spec/QVT/1.1/ Last Access: 2018-03-28

OMG (2011c) OMG Object Constraint Language (OCL), Version 2.3.1 Object Management Group http://www.omg.org/spec/OCL/2.3.1/ Last Access: 2018-03-28

Roth S., Zec M., Matthes F. (2014) Enterprise Architecture Visualization Tool Survey 2014 Software Engineering for Business Information Systems (sebis), Technical University of Munich https://wwwmatthes.in.tum.de/pages/6u8f5ki1t2yz/ Last Access: 2018-03-28

Runeson P., Host M., Rainer A., Regnell B. (2012) Case Study Research in Software Engineering – Guidelines and Examples. John Wiley & Sons, Inc., New York, NY, USA

SAP (2016) SAP Power Designer http://go.sap.com/product/data-mgmt/powerdesigner-data-modeling-tools.html Last Access: 2018-03-28

Sein M. K., Henfridsson O., Purao S., Rossi M., Lindgren R. (2011) Action Design Research. In: MIS Quarterly 35(1), pp. 37–56

Stachowiak H. (1973) Allgemeine Modelltheorie. Springer, Wien, New York

Strecker S., Frank U., Heise D., Kattenstroth H. (2012) MetricM: A modeling method in support of the reflective design and use of performance measurement systems. In: Information Systems and e-Business Management 10(2), pp. 241–276

The Open Group (2009) TOGAF 9 - The Open Group Architecture Framework Version 9 The Open Group https://www.opengroup.org/togaf/ Last Access: 2018-03-28

Tunjic C., Atkinson C. (2015) Synchronization of Projective Views on a Single-Underlying-Model. In: Proceedings of the 2015 Joint MORSE/VAO Workshop on Model-Driven Robot Software Engineering and View-based Software-Engineering. MORSE/VAO '15. ACM, L'Aquila, Italy, pp. 55–58

US Federal Government (2013) FEAF Version 2 https://obamawhitehouse.archives.gov/sites/default/files/omb/assets/egov_docs/fea_v2.pdf Last Access: 2018-03-28

Vassiliadis P. (2009) A survey of Extract–Transform–Load Technology. In: International Journal of Data Warehousing and Mining (IJDWM) 5(3), pp. 1–27

Zachman J. A. (1987) A framework for information systems architecture. In: IBM Systems Journal 26(3), pp. 276–292