# Towards a Language for Enterprise Architecture Documentation and Analysis – Extending the Meta Object Facility

Sabine Buckl, Florian Matthes, René Ramacher, Christian M. Schweda
Chair for Informatics 19
Technische Universität München
E-mail: {buckls,matthes,ramacher,schweda}@in.tum.de

***Abstract*—Enterprise Architecture (EA) management is widely alluded to as an important challenge for today's enterprises. Different approaches from literature target this topic and emphasize on the importance of analyzing the EA as part of the management function. In this respect, different analysis techniques have been proposed, each based on a different representation of the EA in a model. Largely disconnected, different methods for documenting and planning the EA and her evolution have been discussed, leading to object-oriented models for this purpose – the so called *information models*. While these both areas are maturing independently, we experience a low cohesion between them. In particular, the models created for documentation as well as planning on the one hand and the models used for analyses on the other hand differ strongly, when it comes to the underlying meta models and concepts. In this paper, we aim at closing the gap resulting from the aforementioned fact. To do so, we discuss the requirements for a suitable object-oriented meta model and exemplify the requirements with an extension to the OMG's *Meta Object Facility*.**

## I. Introduction & Motivation

Nowadays enterprises have to survive in an ever changing environment. Therefore, the need to be flexible regarding new emerging business demands and adapt quickly to changing market situations is seen as a critical success factor, resulting in special requirements regarding the supporting IT infrastructure. A commonly accepted means to support this alignment between business and IT is enterprise architecture (EA) management. Architecture is, in this context, understood as *the fundamental organization of a system [the enterprise], embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution* [18]. A multitude of approaches to EA management has been developed by researchers [13], [33], [36], [37], practitioners [11], [29], public authorities [9], [10], [35] and tool vendors [27]. Although differing widely regarding the selected focal points most approaches agree on the main goal of EA management, which is guiding organizational change via a managed evolution of the enterprise [28] and enhancing the alignment between business and IT [16], [32]. Thereby the tasks of EA management are a) documenting the current state, b) strategic visioning of future states, c) analyzing EA models of current and future states d) planning and guiding of EA transformations and e) enforcing/realizing planned states.

In oder to support the managed evolution of the enterprise [28], decisions in the context of EA management have to be taken with a holistic perspective, which takes aspects from all layers – strategic via business and organizational, to IT infrastructure – into account (cf. Figure 1). These decisions are typically based on and backed by analysis of EA models, which are used throughout the management process to provide decision support. Using EA analysis, system properties such as availability, performance, flexibility, or operational risk, etc. can be operationalized as metrics and assessed [19]. These metrics can then be used to compare different future planning scenarios and thereby provide decision support for the EA transformation process [24]. Various techniques to perform this kind of analysis have been proposed, e.g. [15], [17], [19] varying e.g. regarding the body of analysis – structural, behavioral, or dynamic behavior – or the time reference – ex ante vs. ex post analysis.

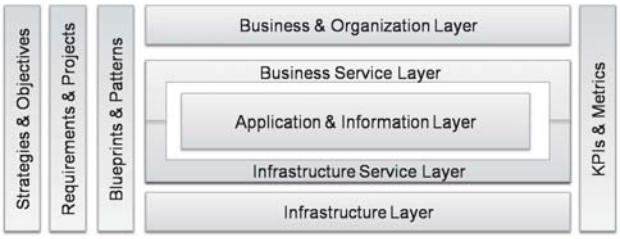


Fig. 1. A holistic perspective on the enterprise [27]

Similar to the multitude of existing approaches regarding analysis techniques as alluded to above, a plurality of corresponding meta models (cf. [3], [23], [26], [35]), which we refer to as *information models* in accordance with [3], defining how to document the constituents of an enterprise on the different layers exist. The existing diversity regarding information models origins on the one hand from the differing purposes these models have been developed for, e.g. Frank et al. propose an information model for indicator-based analysis of EA models [15], while Johnson et al. present an information model supporting dependency modeling for EA analysis [19], and Buckl et al. present an information model capable of storing temporal information to support the managed evolution of application landscapes [5]. On the other hand no commonly

accepted standard information model yet exists [23], which advocates for the idea of such models being enterprise-specific design artifacts.

The decoupled development of information models for EA analysis and EA documentation as referred to above leads to a the utilization of different meta models and concepts for documenting and analyzing the EA. Regarding the close connection of documentation and analysis in the context of EA management, an approach to overcome the aforementioned gap needs to be developed and validated in order to support reusability of gathered information in the documentation task of EA management for EA analysis. This especially applies, as EA analysis and EA documentation are tasks commonly used in conjunction, i.e. the output of the EA documentation serves as input for EA analysis. Our contribution presented in this paper is a first step to the development of an object-oriented meta model suitable for EA documentation and analysis purposes by specifying requirements for such a model and exemplifying the proposed approach by presenting an extension to the *Meta Object Facility (MOF)* of the OMG [30]. The MOF is used in this context as modeling facility on the M2-level (cf. discussions in [30]), which is used to develop information models, i.e. models on M1-level. Further, the MOF is used as the majority of approaches for EA analysis and documentation, as found in literature, is presented in an object-oriented manner.

The remainder of the article unfolds as follows. Based on an overview about the state-of-the-art regarding existing approaches to EA analysis (Section II) and information models (Section III), requirements for an object-oriented meta model, which is suitable for EA analysis and documentation are specified and discussed in Section IV. An approach to address the collected requirements is exemplified by proposing an extension to the MOF of the OMG in Section V. Finally, a critical reflection is given in Section VI, where we discuss our findings and the limitations of the proposed approach and provide and outlook to areas, which need further investigation.

## II. State-of-the-art in EA Analysis Techniques

Decisions taken in the context of EA management are typically based on analysis results of current states of the EA or assessments of planned future states. Thus, different kinds of analysis regarding the EA exist, which differ in respect to coverage of the EA and focal points and the employed techniques used. According to [7] EA analysis techniques can be classified utilizing the following dimensions[1]:

- Three different dimensions regarding the *body of analysis* can be distinct – structure, behavior statistics, and dynamic behavior. Whereas the structural complexity is concerned with the number of constituents and their connections, e.g. the number of interfaces of an application, the behavior statistics take behavioral complexity of the system's constituents into account, e.g. resistance to change of an application [28]. Furthermore, dynamic behavior regards the systems capability to react to certain situations, e.g. the impact of a system failure propagating over time.
- Analysis in the context of EA management, are typically performed on two different types of models – models of the current state or models of future planned states – therefore different *time references* regarding the realization of the object under investigation can be distinct. In the former case, analysis can be based on information, which is measurable as the object of investigation already exists. These analysis are referred to as ex post analysis. In the latter case, the analysis are based on assumptions and predictions about the object of investigation, therefore these analysis are ex ante analysis.
- The formalization of the *analysis technique* varies in the different approaches from expert-based techniques via rule-based to indicator-based ones. The least formal technique, the expert-based one, is the most flexible one, but time-consuming and dependent on the experience and expertise of the executing person. The rule-based technique apply on a more formal level describing either desirable or avoidable architectural constellations, which can be automated. The most formal technique, the indicator-based analysis, allows quantitative assessment of the object under investigation.
- Two types of *analysis concerns* regarding the enterprise function can be distinguished – functional and non-functional concerns. Whereas functional concerns deal with the intrinsic goal of the enterprise itself e.g. production or sales, non-functional concerns take aspects like security, operating, or maintenance costs into account.

We will refer to these classification dimensions, with a special emphasis on the dimensions *body of analysis* and *time reference*, during analyzing the state-of-the-art in EA analysis techniques in the following.

In [8], an approach to perform EA analysis with XML is proposed. Thereby, the article puts a strong emphasis on issues how to represent the EA in a model for performing analyses. This model is rather fine grained, proposing to use *state-machines* to include behavioral descriptions as well as XML elements to represent structural information about the architecture. The time reference in the analyses is not explicitly alluded to, although the structural analyses can be applied both ex-ante and ex-post, while the state-machine based modeling of EA behavior points towards an ex-ante analysis of behavioral aspects. The question how to calibrate behavior models against measured behavior is nevertheless not discussed in the article. Also, the question of the employed analysis technique is not detailed, but indications pointing towards rule-based analysis techniques exist. When it comes to the possible analysis concerns, the paper does neither advocate functional nor non-functional properties of the EA. The approach presented nevertheless can be regarded generic enough to support both types of concerns.

[1] [7] states a fifth dimension concerned with self-referentiality of the analysis approach. We do not regard this dimension to be of interest here, therefore it is neither presented nor used in the following.

| Body of Analysis | structure<br>[29],[8],([15],[20]),[22] | behavior statistics<br>[17],[15],[20] | dynamic behavior<br>[8] |
|---|---|---|---|
| Time Reference | ex-post<br>[29],[8],[17],[15],[20],[22] | ex-ante<br>([8]),[17],[15],[20],([22]) | |
| Analysis Technique | expert-based<br>[29] | rule-based<br>[8],[22] | indicator-based<br>([29]),[17],[15],[20] |
| Analysis Concern | functional<br>[29],([8]),[22] | non-functional<br>([29],[8]),[17],[15],[20],[22] | |

TABLE I
SUMMARIZING CLASSIFICATION OF EA ANALYSIS APPROACHES

A more quantitative approach to EA analysis is proposed in [17], where exemplary indicators for analyzing EA behavior are provided. These indicators are developed to provide aggregated information on the behavior of certain EA artifacts, i.e. to compute behavior statistics. The analysis model presented in the approach integrates into the ArchiMate architecture description language[2], whose concepts are augmented with additional properties reflecting non-functional aspects of the EA, as e.g. the execution or completion time for a business process. The time reference in the analysis is discussed alongside the question of the quantitative input for the analysis model. In particular, the authors highlight that measuring the behavior of an existing system can provide valuable input, but also rise the question of reproducible circumstances for the measurements. For systems, which are still to be developed, estimates for properties, e.g. based on comparable architectures, are noted as possible source for quantitative information. Based on this information performance measures for the (planned) system are derived analytically, i.e. the values for descriptive indicators are computed.

Two prominent approaches for EA analysis are presented in [15] and [20], respectively. These approaches vary significantly concerning their origins and concerning the concepts, which they employ. Concerning the classification dimensions, both approaches are nevertheless quite similar. The two approaches support both ex-ante and ex-post analyses of enterprise systems represented in EA models. In particular, [15] emphasizes the importance of an indicator system, which is used for analyses there, as means to support communication in an enterprise. The focus of both approaches lies on statistic information arising from the behavior of the enterprise system, although to a limited amount also structural aspects of the EA are analyzed. Beside the indicator-based analysis technique, a prominent difference exists: the approach of [20] aims at the development of single indicators, while [15] goes even further. In particular, the latter approach seeks to develop integrated and consistent indicator systems. Regarding the analysis concern, both approaches focus on non-functional requirements, with the approach of [15] centering on more economic indicators, whereas [20] puts an emphasis on *classical* non-functional aspects, such as availability.

The topic of EA analysis is also discussed in [29]. There an emphasis is put on the utilization of EA models, which have been created during other EA management activities. The models discussed therein reflect structural aspects of the EA, thus limiting the body of analysis on such aspects. Furthermore, existing enterprise systems reflected in their EAs are analyzed according to different concerns. The majority of these concerns is functional, e.g. homogeneity of the application landscape or the interdependencies between the business applications are considered. Complementing, two non-functional concerns, namely *costs* and *benefits*, are alluded to as typical EA analysis concerns. Nevertheless, the measures for those costs and benefits remain on a rather abstract level. The different analysis concerns are addressed by different techniques, of which the majority is expert-based, i.e. utilizes specific viewpoints to present architecture information to an enterprise architect, who informally assesses e.g. the level of homogeneity. When it comes to the non-functional concerns, the predominance of the expert-based analysis is broken in favor of a few quantitative indicators, which are especially used to operationalize benefits.

Agreeing with the overall understanding of EA analysis as proposed in [29] (cf. [2]), a dedicated support of *impact analyses* is discussed in [22]. In particular, the EA is thereby understood as a directed graph reflecting the structure of the enterprise system. On this graph, rule-based analyses are performed to assess and evaluate the transitive impact of an EA constituent, e.g. in cases of failure. Thereby, the analyses are applied to current architectures, although the proposed method is not limited and could hence also be applied on planned architectures. Regarding the analysis concerns, the approach does not make assumptions, i.e. it can handle both functional and non-functional ones. The following two analysis questions taken from [22] exemplify this: *Which business applications are used during the creation of a selected product?* and *Which applications fail, if a certain server fails?*.

The results of the state-of-the-art analysis are summarized in Table I. Summing up, it can be stated, that the above presented approaches differ widely regarding the focus set in the analysis. Nevertheless, their description represent a good starting point to specify requirements for a meta model suitable for EA documentation and analysis as presented in Section IV.

## III. STATE-OF-THE-ART IN EA INFORMATION MODELING

In the preceding section, we discussed the plurality of analysis techniques that are applied in the context of EA management. This plurality is also reflected in the manifold

[2]For more information see http://archimate.telin.nl.

types of EA models, the techniques rely on. This section is dedicated to these models, in particular to their meta models, which are subsequently called *information models*. Preparing the subsequent discussions, it has to be noted that irrespective the prominence of EA management as research field, no standard information model for EA management yet exists. In contrast, the different EA management approaches from literature bring along their own information models varying greatly in respect to both coverage and level of detail. To prevent misunderstands, we apply a writing convention to the names of information model concepts on the one hand and corresponding meta level concepts on the other hand. An information model concept is hence denoted as follows `Concept`, while a meta level concept of the same name was denoted as Concept.

A language for EA descriptions is introduced in [21], as a language following the paradigm of *object-orientation*. Therefore, the basic concepts Thing and Relation are introduced, from which subconcepts specific to the EA domain are derived by refinement. Due to this restriction, attributes for the modeling concepts cannot be supplied by means of the language, although the respective instance models advocate for the existence of a name attribute. Centrally, the language builds on a conceptual framework, partitioning the EA along three different *aspects* and three different *layers*. The topmost layer of the framework is concerned with *business* concepts such as business processes, which are supported by *application* layer concepts as applications. The latter themselves rely on *infrastructure* layer concepts as infrastructure components. The aspects establish a partition of the EA concepts in *structure*, *information*, and *behavior* concepts. This partition further is aspect-independent, such that generic concepts for structure (e.g. *services*), information (e.g. *messages*), and behavior (e.g. *events*) are introduced. These concepts are additionally connected by generic relationships. By refinement, layer specific concepts are derived, but no refinement applies to the relationships. This modeling technique introduces additional flexibility to the information model of [21], as it allows to omit e.g. application layer concepts, if no information is present. This flexibility nevertheless does not come without cost, but can lead to imprecisions and inconsistencies, if application layer concepts are amended later.

The *core business meta model* presented in [34] is according to its intended usage context an information model for EA modeling. This information model is described using a subset of the modeling facilities as presented by the UML [31], namely classes, associations, aggregations, and association classes. The core business meta model prominently focuses on describing business concepts, such as processes and products, and relating them to supportive concepts as business applications. Nevertheless, the overall perspective is strongly business centric. Due to the decision of only using a subset of the UML, powerful concepts, such as properties, for further detailing the described EA are omitted. Also, concepts for ensuring model consistency, such as multiplicities, are not present in the core business meta model. Nevertheless, they are alluded to in [34], where they are described to be enterprise-specific and hence are subject to introduction in an model adaptation process complementing model usage. The authors of [34] further describe that during the adaptation mechanism the using enterprise can decide to set aside some concepts or to replace association classes by simple associations, if the corresponding more detailed information is not needed. Nevertheless, no statements are made on the exact procedure of the adaptation mechanism, nor on a relationship between the adapted model and the analysis techniques applied upon.

The *multi-perspective enterprise modeling* (MEMO) method [13] emphasizes on the specificity of EA models in respect to their stakeholders. Therefore, different aspects of the EA, e.g. the IT aspect or the business process aspect, are each incorporated into a single modeling language, bringing along a dedicated information model for the aspect. Using these languages, the enterprise can be described from different perspectives which correspondingly focus on selected aspects of the enterprise. Complementing the special purpose modeling language, an integrative meta language, called *MEMO meta modeling language* (MML) [14] exists. This language introduces a common object-oriented meta model for the different languages' information models. This meta model shares multiple characteristics and concepts of the UML [31], beginning with the graphical notation for the language concepts. The most prominent exception of standard UML is the concept of the intrinsic feature, which breaks the strict *type-instance* layering of the UML. With an intrinsic feature, it is e.g. possible to also model properties, not only meta properties, on the meta level. These properties are used normally on the instance-level, thus omitting further specification demands on the intermediary information model level. This resembles the notion of the *clabject* and the *potency* as discussed in [1], although only potencies of **1** and **2** are supported via the intrinsic features.

In [4], [12] a pattern-based approach to EA management is presented. Therein, so called *I-patterns* (short for *information model patterns*) are introduced, that supply the information models for describing the EA. The I-patterns are therein understood as best-practice solutions to recurring problems in EA modeling, which themselves are connected to recurring problems in EA management, the so-called *concerns*. By applying the idea of patterns to this context, the information models can be kept small and narrowly focused to the problem, they are meant to address. In this respect, the approach continues the basic intention of MEMO to a smaller scale, i.e. where languages in MEMO represent a distinct perspective, the concerns decompose the perspective to typical problems. The I-patterns utilize the meta concepts as provided by the OMG's MOF, more precisely the *essential* parts thereof as subsumed in the EMOF. Additionally, selected patterns as presented in [25] employ the *Object Constraint Language* (OCL) to provide additional information, e.g. to specify rules and indicators for analyzing EA models instantiating the corresponding information model. Nevertheless, the employment of both EMOF and OCL has some subtle complexities associated,

which are not discussed in detail by the approach. At first, a language binding of OCL to EMOF does not exist, i.e. OCL expects language features that are not present in EMOF such that the semantics of OCL-expression is not always clear. Secondly, the I-patterns present solutions for specific EA management problems. A holistic EA management endeavor would hence have to utilize several of them in an integrated manner. For this integration question, the standard mechanisms of the EMOF do not provide a comprehensive solution. Mechanisms for merging packages[3] as introduced in the UML infrastructure [31] might be helpful in this context, but are not without difficulties regarding their formal semantics.

The above list of approaches to EA information modeling is by far not complete but does nonetheless provide a good overview on the state-of-the-art by detailing representative and prominent approaches in this area.

## IV. Requirements for an EA Meta model for Supporting EA Analyses

Having discussed the state-of-the-art in the fields of EA analysis and of EA information modeling, it quickly becomes obvious that only weak links between these fields exist. Especially, the way to represent EAs as instances of EA information model concepts differs strongly from the EA representations as used by the different analysis techniques. An exception in this context is the approach of the Score-ML that combines the object-oriented *multi-perspective enterprise modeling* (MEMO) [13] with a technique to augment the object-oriented models with business indicators. The requirements that led to this combination technique are discussed in [15] and are revisited below.

The Score-ML method supports and enforces the design of comprehensive and consistent, i.e. non-contradicting and non-conflicting indicator systems. Further, the method allows to support the user with documentation helpful for interpreting the indicators and understanding the context, in which the indicators apply. In this vein, the rationale for an indicator can also be supplied. The indicators should be represented on different levels of abstraction for different stakeholders. Finally, the method of the Score-ML is intended to foster the creation of a software system implementing the corresponding indicator systems.

Concluding, it can be said, that the above requirements strongly emphasize the consistent usage of the analysis model and elaborate on ways to ensure this consistency by construction. Therefore, interconnections are introduced to describe that (and how) an indicator is influenced by related architectural indicators. In contrast, details of setting up the analysis model and calibrating the model are only rudimentarily alluded to. Applying the classification dimensions *body of analysis* and *time reference* as introduced in Section II, this quickly becomes obvious. The analysis technique claims to be applicable for both ex-ante and ex-post analyses, although especially with the former, the situation is not that clear. Looking at analyses of the EA behavior (either statistic or dynamic), one quickly recognizes that the analysis technique of the Score-ML does not comprise means for supporting ex-ante analyses of the behavior. Such analyses can only be built on estimations of the behavior, which are per se only feasible regarding behavior statistics.

With the above discussions in mind, we complement the *definitional* relationships between architectural properties and indicators as introduced by [15] with *cause-effect* relationships. The latter concept should be used to incorporate behavioral aspects into a model of the EA. To detail on the distinction between the two types of relationships, we subsume them as two requirements as follows:

- R1 Not all indicators used for EA analysis are directly observable architectural properties. For the indicators, which are not directly observable, the information model must be augmented with information on their definition, i.e. on the architectural properties that these indicators depend on.
- R2 Certain architectural properties might exist, which do not change their values independently, but are related to each other in a way that a change of one property causes effects on the values of dependent properties. This change is therein not the immediate consequence of an existing definitional relationship.

Exemplifying the above requirements, one could think of the property `available` of a server. It can either take the value `true` or `false` at a given point in time; further, the value of the property can be determined by observation. Presenting the time-series of the property `available` might not be sensible during EA analysis; in contrast, a derived property `availability` is computed from the time-series. In terms of the above relationship concepts hence a definitional relationship between `available` and `availability` would exist. Continuing the example, think of an e-mail service installed on the aforementioned server. This service would also have a property `available` associated in the EA model. Obviously the value of this property would depend on the value of the server's `availability` property. Put in other words, the e-mail service cannot be available, if the underlying server is not. This kind of relationship is in contrast no definitional one, but is intrinsic to the system under consideration, i.e. it is a cause-effect relationship.

Retrospecting requirements **R1** and **R2**, one notices the repeated usage of the term *observable*. This should not surprise anyone, as analyzing architectures is mostly to be related to observing them, especially if behavioral aspects are considered. Consistent with a central idea of the *architecture theory diagrams* (cf. [20]), we establish a corresponding requirement (**R3**) for our approach. Additionally, it has to be noted that in our above considerations, no assumptions on the details of a relationship between architectural properties have been made. In particular, the exact type of the relationship as expressed e.g. via a computation rule remains open. We do not assume this to be a drawback of the approach, as the

[3] One could assume an I-pattern to be a package.

relationships are primarily intended to express *that* a relation exists. Nevertheless, the approach must provide a method to complement this qualitative relationship information with computable rules. Requirement **R4** formulates this.

R3 Architectural properties can be distinguished into observable properties and latent ones. The values of the latter properties are hence not determined by observation. The distinction between the types of properties is not fixed, but depends on the actual decision of what to observe.

R4 Definitional and cause-effect relationships only express that two architectural properties are related. In this respect, it must be possible to complement these relationships with computable rules expressing the actual type of relation, i.e. providing an operationalization.

In terms of the running example from this section, requirement **R3** expects us to mark the `available` properties of the server and the business application as observable. The `availability` of the server would in contrast be marked as latent. In the vein of **R4**, the definitional relationship between `available` and `availability` is operationalized by a moving average function. The cause-effect relationship between the two `available` properties is complemented by a conditional probability table, indicating that the business application can only be available, if the underlying server is.

A final requirement (**R5**) can be derived by revisiting the notion of the architecture theory diagrams from [20]. There, intermediary properties are introduced, i.e. properties that are neither relevant indicators nor observable architectural properties. In particular, the intermediary properties only serve as means to simplify the formulation of the dependencies. In an object-oriented technique for describing relationships between properties, one could also expect that such intermediary properties exist. This would especially be the case, if no direct association between the class with the *source* property and the class holding the *target*, i.e. the related, property existed. We nevertheless, regard the introduction of such intermediary properties as unnecessary complification of the model that is detrimental to the readability of the model. Therefore, the following requirement formulates our demand to support transitive property relationships without using intermediary properties:

R5 Definitional and cause-effect relationships cannot only relate architectural properties in the same class. In the case, where the source and the target property of a relationship are owned by different classes, the relationship must specify the association(s) that (transitively) link the two owning classes.

## V. An Extension to EMOF for Supporting EA Documentation and Analyses

Based on the requirements specified in the preceding section, we develop a meta model for supporting EA documentation and analyses. As a starting point, we take the object-oriented meta model of the OMG's EMOF [30] and provide extensions to ensure that the model fulfills the requirements. Thereby, build on preliminary results in this respect from [6]. Preparing our subsequent discussions, we introduce a writing convention to prevent mixing up concepts on meta level (M2) and concepts from the information model (M1) – meta level concepts are added the prefix *meta-* in the following, while information level concepts are denoted without prefix. The type-setting convention as introduced above further applies here.

Requirement **R3** demands that architectural properties are distinguished into observable and latent ones. Consistent with the findings from Section III, we assume that those properties are mostly reflected in the information model by Property concepts, i.e. instances of the meta-class Property. In this respect, **R3** is quite simple to fulfill by introducing an additional meta-property to the respective meta-class. This meta-property isObservable is of boolean type and allows to indicate, whether the respective property can be observed or not. Complementing the extension of the abstract syntax of the EMOF via an additional meta-property, we further propose to use a graphical symbol to indicate the status of the isObservable meta-property for a property from the information model. In a simplistic case, one could add the term {`observable`}, where the meta-property is true.

Two aspects of adding the meta-property to the concept Property remain to be discussed at this point. At first, one could have used the standard meta-property isDerived instead. Secondly, meta-properties of EMOF are also used to model relationships between meta-classes – what is the implication of observable in this context. Discussing this second aspect first, the fact that also relationships can be marked as observable seems to be quite consistent with the understanding of architectural documentation. In particular, this relates to the first question regarding the distinction between derived and observable. Derived properties in the understanding of the MOF do not bear additional information in their values, as those can be computed from the values of the other non-derived properties. In contrast, the observable meta-property does not make assumptions on the values of some properties to be computed (they could have been estimated instead). The meta-property only indicates, that the value of a respective property is gained by observation of the enterprise under consideration, i.e. is *manifest*.

The AttributeRelationship as proposed and exemplified in [6] provides a suitable technique for expressing that two properties relate. Nevertheless, the concept cannot be used to specify the type of relationship, i.e. to distinguish between definitional and cause-effect relationships. In this respect, we introduce the generic and abstract PropertyDependencyRelationship concept, from which dedicated concepts for the different relationship types are derived by specialization. Before describing these specialized classes, we elaborate on the influenceType meta-property of the base class. This meta-property allows to optionally specify, what kind of influence the related property receives from the source property. One can abstain from providing a value, if the exact type of influence is

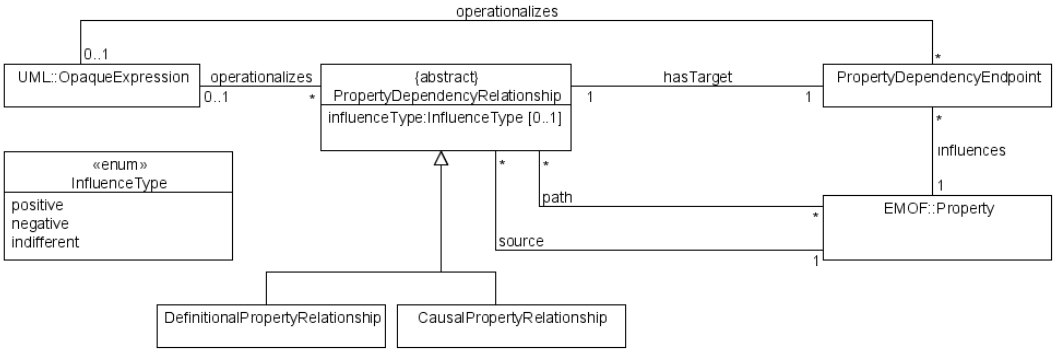


Fig. 2. Meta model extension for modeling EA analyses

unknown, or can select a value from the three choices positive, negative, and indifferent. Thereby, it can be expressed that the related property behaves similar to the source property (positive) or the exact opposite (negative), or nothing can be said about the relative behavior (indifferent)[4].

Requirement **R5** demands the PropertyDependencyRelationship meta-class to have a meta-property for supplying the *path* of associations connecting the class owning the source with the class owning the target architectural property. Therefore, the dependencyPath meta-property is introduced as a multi-valued one, referencing properties describing a (transitive) link from the source to the target class. To ensure, that this kind of path modeling leads to sensible models, we further must introduce some additional constraints, which are detailed at the end of the section with the respective UML-model of the provided extension.

Making explicit the actual type of dependency represented (as demanded by **R4**) by the respective instances, the generic concept allows to optionally supply an expression (type OpaqueExpression as defined in the UML infrastructure). This expression provides the actual rules for computing the dependency, i.e. for determining the target value based on the corresponding source values. As multiple dependency relationships can target the same target value, an additional concept PropertyDependencyEndpoint is introduced. Via this concept, it is possible to supply another expression for combining the values supplied by the dependency relationships. The type OpaqueExpression is – according to its definition in the UML infrastructure – a base type for expresions over object-oriented models. Different languages can subclass the concept to introduce their own hierarchy of expressions. The OCL, for example, introduces the OCLExpression concept that inherits from OpaqueExpression; therefore, OCL expressions can be supplied to detail the actual relationship between the source and the target property. This possibility should nonetheless not be mistaken: we do not advocate the use of the OCL for detailing property dependency relationships. This language is not especially designed towards the purpose and hence leads to cumbersome expressions mimicking a relationship, which can be expressed more concisely in a different language – we even show this problem in our running example below.

From the general relationship concept more specific concepts representing the definitional and cause-effect relationship respectively are derived by inheritance. Thereby, requirements **R1** and **R2** are fulfilled, respectively. The class DefinitionalPropertyRelationship expresses, that the target property is defined on the source properties. In this respect, the target property does not have a dedicated value assigned, but the value is computed from the source properties via the supplied expressions. In contrast, the class CausalPropertyRelationship expresses, that the target property has a dedicated value, which is influenced by the value of the corresponding source property. Therefore, the supplied expression is different in its *signature*, i.e. it does not only have one input value (the source value) but also takes the current target value as input value. The expression then evaluates to the new target value, i.e. a target value of a later period in time. In this vein, causal property relationships can be used to model behavioral aspects of the system under consideration. In particular, a notion of time introduced into the model, although the time reference, e.g. the length of the periods, is not explicitly alluded to.

Complementing the above discussions on the respective model concepts, we provide the corresponding meta model extension for EMOF in Figure 2. Two constraints apply for all PropertyDependencyRelationship instances. At first, the source property and the target property, as associated via the corresponding PropertyDependencyEndpoint instance, must not be identical. Secondly, the path must connect the source must connect the class owning the source property and the one owning the target property in an acyclic manner.

## VI. Conclusion and Outlook

In this paper, we discussed the state-of-the-art in EA analysis and in EA modeling. In this discussion, we could show that the fields develop independently. This results in the absence of a common meta model, suitable for analyzing the EA on the one hand and modeling, especially documenting and planning, the EA on the other hand. To close the resulting gap, we spawned the idea of an integrated meta model for

[4]One can identify the last relationship type with a *knowing null* different from the *unknown type* case.

EA modeling and EA analysis. Taking an object-oriented modeling technique for EA documentation and planning as given, we specified additional requirements an object-oriented meta model had to fulfill to adequately integrate EA analysis techniques. In a final section, we showed how these requirements can be addressed in an extension to the OMG's EMOF.

The extended meta modeling facility, presented in response to the experience lack of support, has yet not been applied in practice. Initial projects are under way and show the applicability, but are far from a comprehensive validation of the proposed meta model. In this context, initial findings reveal that adequate tool support would be very beneficial in applying the modeling technique in practice. Furthermore, the utilization of the OpaqueExpression concept to allow for operationalizing the property relationships introduced an additional complexity to the meta model, whose full potential was yet not leveraged. In particular, as only OCLExpression concepts are currently used to implement the operationalization, the intended decoupling of qualitative dependency modeling and dependency operationalization is not fully realized. This opens to an interesting area of future research, as object-oriented formalizations of other computation models exist. It would hence be both interesting and beneficial to see, how these models could be integrated to the analysis formalism via appropriate inheritance from OpaqueExpression. In this respect, challenges regarding analysis models that are non isomorph to the documentation models might arise.

A even more general question for future research is concerned with the basic object-oriented meta modeling facility herself. Having shown in the field of EA analysis, that basic object-oriented facilities are not sufficient, one might rise the question, if there were activities in the context of EA management which advocate additional extensions for the meta modeling facilities. Especially the MML meta language presented in [14] supports this idea, although we discussed that this facility lacks specific support for EA analysis concepts (cf. Section III). With this general question is background, we expect the area of meta modeling language for EA information modeling to be an interesting field for future research.

## REFERENCES


[1] C. Atkinson and T. Kühne. Reducing accidental complexity in domain models. *Software and Systems Modeling*, pages 345–359, 2007.

[2] T. Bucher, R. Fischer, S. Kurpjuweit, and R. Winter. Analysis and application scenarios of enterprise architecture: An exploratory study. In *Tenth IEEE International Enterprise Distributed Object Computing Conference (EDOC 2006), 16-20 October 2006, Hong Kong, China, Workshops*, IEEE Computer Society, Washington, DC, USA, 2006.

[3] S. Buckl, A. M. Ernst, J. Lankes, and F. Matthes. Enterprise Architecture Management Pattern Catalog (Version 1.0, February 2008). Technical report, Chair for Informatics 19 (sebis), Technische Universität München, Munich, Germany, 2008.

[4] S. Buckl, A. M. Ernst, J. Lankes, K. Schneider, and C. M. Schweda. A pattern based approach for constructing enterprise architecture management information models. In *Wirtschaftsinformatik 2007*, pages 145 – 162, Universitätsverlag Karlsruhe, Karlsruhe, Germany, 2007.

[5] S. Buckl, A. M. Ernst, F. Matthes, and C. A. Schweda. An information model for managed application landscape evolution. *Journal of Enterprise Architecture (JEA)*, 5(1), pages 12 – 26, 2009.

[6] S. Buckl, U. Franke, O. Holschke, F. Matthes, C. M. Schweda, T. Sommestad, and J. Ullberg. A pattern-based approach to quantitative enterprise architecture analysis. In *15th Americas Conference on Information Systems (AMCIS)*, San Francisco, USA, 2009.

[7] S. Buckl and C. M. Schweda. Classifying enterprise architecture analysis approaches. In: 2nd IFIP WG5.8 Workshop on Enterprise Interoperability (IWEI'2009), Valencia, Spain, 2009.

[8] F. S. de Boer, M. M. Bonsangue, J. Jacob, A. Stam, and L. van der Torre. Enterprise architecture analysis with xml. In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences (HICSS 2005)*, volume 8, page 222b, IEEE Computer Society Press, Los Alamitos, CA, USA, 2005.

[9] Department of Defense (DoD) USA. DoD Architecture Framework Version 1.5: Volume I: Definitions and Guidelines. http://www.defenselink.mil/cio-nii/docs/DoDAF_Volume_I.pdf (cited 2009-06-30), 2007.

[10] Department of Defense (DoD) USA. DoD Architecture Framework Version 1.5: Volume II: Product Descriptions. http://www.defenselink.mil/cio-nii/docs/DoDAF_Volume_II.pdf (cited 2009-06-30), 2007.

[11] G. Dern. *Management von IT-Architekturen (Edition CIO)*. Vieweg, Wiesbaden, Germany, 2006.

[12] A. Ernst. Enterprise architecture management patterns. In *PLoP 08: Proceedings of the Pattern Languages of Programs Conference 2008*, Nashville, USA, 2008.

[13] U. Frank. Multi-perspective enterprise modeling (memo) – conceptual framework and modeling languages. In *Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS 3003*, pages 1258–1267, 2002.

[14] U. Frank. The memo meta modelling language (mml) and language architecture (icb-research report). Technical report, Institut für Informatik und Wirtschaftsinformatik, Duisburg-Essen, Germany, 2009.

[15] U. Frank, D. Heise, H. Kattenstroth, and H. Schauer. Designing and utilising business indicator systems within enterprise models – outline of a method. In *Modellierung betrieblicher Informationssysteme (MobIS 2008) – Modellierung zwischen SOA und Compliance Management 27.-28. November 2008*, Saarbrücken, Germany, 2008.

[16] J. C. Henderson and N. Venkatraman. Strategic alignment: leveraging information technology for transforming organizations. *IBM Systems Journal*, 38(2-3), pages 472–484, 1999.

[17] M.-E. Iacob and H. Jonkers. Quantitative analysis of enterprise architectures. In D. Konstantas, J.-P. Bourrières, M. Léonard, and N. Boudjlida, editors, *Interoperability of Enterprise Software and Applications*, pages 239–252, Springer, Geneva, Switzerland, 2006.

[18] International Organization for Standardization. ISO/IEC 42010:2007 systems and software engineering – recommended practice for architectural description of software-intensive systems, 2007.

[19] P. Johnson, E. Johansson, T. Sommestad, and J. Ullberg. A tool for enterprise architecture analysis. In *11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007), 15-19 October 2007, Annapolis, Maryland, USA*, pages 142–156. IEEE Computer Society, 2007.

[20] P. Johnson, L. Nordström, and R. Lagerström. Formalizing analysis of enterprise architecture. In *Interoperability for Enterprise Software and Applications Conference*, page 10, Springer, Bordeaux, France, 2006.

[21] H. Jonkers, R. van Burren, F. Arbab, F. de Boer, M. Bonsangue, H. Bosma, H. ter Doest, L. Groenewegen, J. Scholten, S. Hoppenbrouwers, M.-E. Iacob, W. Janssen, M. Lankhorst, D. van Leeuwen, E. Proper, A. Stam, L. van der Torre, and G. van Zanten. Towards a language for coherent enterprise architecture descriptions. In *7th International Enterprise Distributed Object Computing Conference (EDOC 2003)*, IEEE Computer Society, Brisbane, Australia, 2003.

[22] S. Kurpjuweit and S. Aier. Ein allgemeiner Ansatz zur Ableitung von Abhängigkeitsanalysen auf Unternehmensarchitekturmodellen. In *9. Internationale Tagung Wirtschaftsinformatik (WI 2007)*, pages 129–138, Österreichische Computer Gesellschaft, Wien, 2009.

[23] S. Kurpjuweit and R. Winter. Viewpoint-based meta model engineering. In M. Reichert, S. Strecker, and K. Turowski, editors, *Enterprise Modelling and Information Systems Architectures – Concepts and Applications , Proceedings of the 2nd International Workshop on Enterprise Modelling and Information Systems Architectures (EMISA'07), St. Goar, Germany, October 8-9, 2007*, LNI, pages 143–161. GI, 2007.

[24] J. Lankes. *Metrics for Application Landscapes – Status Quo, Development, and a Case Study*. PhD thesis, Technische Universität München, Fakultät für Informatik, Munich, Germany, 2008.

[25] J. Lankes and C. M. Schweda. Using metrics to evaluate failure propagation and failure impacts in application landscapes. In *Multikonferenz Wirtschaftsinformatik*, GITO-Verlag, Berlin, Germany, 2008.

[26] M. Lankhorst. Introduction to enterprise architecture. In *Enterprise Architecture at Work*, Springer, Berlin, Heidelberg, New York, 2005.

[27] F. Matthes, S. Buckl, J. Leitel, and C. M. Schweda. *Enterprise Architecture Management Tool Survey 2008*. Chair for Informatics 19 (sebis), Technische Universität München, Munich, 2008.

[28] S. Murer, C. Worms, and F. Furrer. Managed evolution. *Informatik Spektrum*, 31(6) pages 527–536, 2008.

[29] K. D. Niemann. *From Enterprise Architecture to IT Governance – Elements of Effective IT Management*. Vieweg+Teubner, Wiesbaden, Germany, 2006.

[30] OMG. Meta Object Facility (MOF) core specification, version 2.0 (formal/06-01-01), 2006.

[31] OMG. UML 2.2 Infrastructure Specification (formal/2009-02-04), 2009.

[32] W. Ross, Jeanne, P. Weill, and C. Robertson, David. *Enterprise Architecture as Strategy*. Harvard Business School Press, Boston, Massachusetts, USA, 2006.

[33] A.-W. Scheer. *ARIS – Modellierungsmethoden, Metamodelle, Anwendungen*. Springer, Berlin, Germany, 4 edition, 2001.

[34] H. Österle, R. Winter, F. Hoening, S. Kurpjuweit, and P. Osl. Der St. Galler Ansatz des Business Engineering: Das Core Business Metamodel. *Wisu – Das Wirtschaftsstudium*, 2(36), pages 191–194, 2007.

[35] The Open Group. TOGAF "Enterprise Edition" Version 9. San Diego, USA, http://www.togaf.org (cited 2009-07-10), 2009.

[36] R. Winter and R. Fischer. Essential layers, artifacts, and dependencies of enterprise architecture. In *EDOC Workshop on Trends in Enterprise Architecture Research (TEAR) within the Tenth IEEE International EDOC Conference (EDOC 2006)*, page 30. IEEE Computer Society, 2006.

[37] J. Zachman. Enterprise architecture: The issue of the century. *DATABASE PROGRAMMING AND DESIGN*, pages 1–13, 1997.