A meta-language for Enterprise Architecture analysis

Sabine Buckl[1], Markus Buschle[2], Pontus Johnson[2], Florian Matthes[1], and Christian M. Schweda[1]

[1] Chair for Software Engineering of Business Information Systems (sebis),
Technische Universität München,
Boltzmannster. 3, 85748 Garching,, Germany
{sabine.buckl,matthes,christian.m.schweda}@mytum.de
[2] Industrial Information and Control Systems,
KTH Royal Institute of Technology,
Osquldas v. 12, SE-10044 Stockholm, Sweden
{markusb,pontus}@ics.kth.se,

Abstract. Enterprise Architecture (EA) management is a commonly accepted instrument to support strategic decision making. The objective of EA management is to improve business IT alignment by making the impact of planned changes explicit. The increasing interconnectivity of applications with other applications and with business processes however makes it difficult to get a complete view on change impacts and dependency structures. This information is nevertheless required to support decision makers. Current meta-languages proposed for the context of EA management provide only limited support for modelling qualitative and quantitative dependencies.
In this paper we propose a meta-language, which builds on the Meta Object Facility (MOF). This meta-language specifically accounts for the requirements of EA analysis. We discuss existing meta-languages from the field of EA management and related areas against these requirements. Building on the standard of the OMG, we present an extension of MOF designed to support EA analysis. The theoretic exposition of the extension is complemented by an example illustrating the applicability of the presented meta-language.

1 Introduction and motivation

Today, the strategic management of the Enterprise Architecture (EA) is a commonly accepted instrument of modern enterprises. EA is used to keep up with the increasing demand for flexible IT support and the overall managed evolution of the enterprise. Therein EA analysis [1] is a means of providing decision support throughout the management process by making the impact of planned projects explicit. In the complex and interwoven system "enterprise", local changes to one artifact, e.g. a business process or a business application, might have unforeseen global consequences and potentially detrimental impacts on related artifacts.

Therefore, it is difficult to get a complete view on change impacts and dependency structures, this is nevertheless required to support decision making.

With regards to the increased interest from practitioners, different approaches to EA management have been developed in academic research [2–4], by practitioners [5, 6], standardization bodies [7, 8], and tool vendors [9]. These approaches provide frameworks, methods, and models used in the design of an EA management function. Thereby, the models typically focus on *structural* aspects of the EA, such as dependencies between business processes and business applications. Decisions about the future of the EA are made based on the analysis of these dependencies and plans for a managed evolution are created (cf. [10]).

Input to the aforementioned analysis are (parts of the) architectural descriptions of the EA. A variety of modelling techniques and complementing information models underlying these architectural descriptions exists. We understand an *information model* in line with Buckl et al. in [11] as "a model which specifies, which information about the ea, its elements and their relationships should be documented, and how the respective information should be structured". Existing information models differ widely with respect to the concepts that they employ as well as the coverage of the EA that they aim at. This diversity backs the assumption of researchers in the EA management domain, that information models represent organization-specific artifacts (cf. [11, 12]).

The models described above have in common that they are implicitly or explicitly based on meta-languages. In the context of EA management a widely used meta-language for EA information modelling is the general purpose modelling language *unified modelling language* (UML) [13]. UML and other meta-languages proposed for the context of EA management provide only limited support for modelling quantitative and qualitative dependencies. In particular the latter are of interest in the context of EA management. They express that *some kind* of dependency exists between attributes, while not having to indicate of *what kind* this dependency is. The aforementioned problem statement that we seek to address can be summarized as follows:

> How does a meta-language look like that supports EA analysis by enabling the user to model different types of dependencies and attributes?

In this paper, we propose a meta-language that builds on the *Meta Object Facility* (MOF) [14] and specifically accounts for the requirements of EA analysis. This approach was chosen as MOF represents the basis of UML a frequently used meta-language for the context of EA management [15]. We prepare the exposition of our solution by eliciting requirements that such a meta-language should fulfill in Section 2. Based on the requirements we revisit related work from the EA management domain and related fields in Section 3. In Section 4 we discuss an extension of *essential* MOF (EMOF) as a meta-language for EA analysis. Our meta-language addresses the afore elicited requirements by providing means for specifying, that the values of certain attributes are dependent on other attributes' values without creating the need to provide computable dependency rules. The theoretic exposition of our meta-language is complemented by an example demonstrating applicability of the presented solution in Section 5. Finally, we conclude the paper with a discussion on further areas of research.

2 Requirements on the meta-language

This section presents requirements on the meta-language we propose. In the previous chapter we already described the fact that the language should extend MOF. This implies that the meta-language should feature classes that can be linked via relations, and are characterisable through attributes. These characteristics are taken from MOF right away. Besides those requirements several more prerequisites need to be considered with regards to the EA domain. These needs have been derived from [16] and [17], which both explicitly state requirements on an expressive meta-language for EA analysis. Especially the characteristics of attributes are discussed in both previously mentioned publications.

It has been stated in [16] and [18] that EA models are likely to become (partly) out-dated after short time periods. The fact that a certain application might for example be phased-out and replaced, is not necessarily resulting in an immediate update of the EA model. If such a discrepancy between the status quo in the real world and the model is not eliminated it is likely that the architectural model does not any longer represent the configuration that is in use. This phenomenon has also been identified in [17] and described as empirical uncertainty.

An expressive meta-language is required to be able to capture whether a model is likely to reflect the setting it is meant to describe (**Requirement R1**). This means that a meta-language should allow to annotate each modeled class with a probability that this modeled entity still is in use (**Requirement R1.1**). To consider only the entities of a model is however not sufficient, moreover it needs to be considered whether the modeled concepts are still related to the same entities i.e. whether they still interact in the same manner with their environment as it was described. Here one could think of a modeled application that in comparison to when it was described in a model nowadays supports a billing business process instead of a order handling process. This means that the application's relation to the order handling process does not exist any longer. The implication is that not only entities but also their relations should be describable with regards to their existence (**Requirement R1.2**). Therefore both entities and their relations need to be equipped with an additional build-in *existence* attribute reflecting the probability that the concept is still employed and collaborating with the artifacts it is related to.

A second aspect of relevance that was identified targets the expressiveness of the attributes offered by the meta-language. While typical general purpose modelling languages as UML can be used to model discrete phenomena, e.g. the number of components of an application system, creating EA models is often accompanied by uncertainty about the actual value of an attribute. The build-in data-types offered by UML and thereby MOF, for example *Boolean* or *Integer*, can be used to describe properties of classes, e.g. a *Boolean* attribute can be either *true* or *false* but never in an intermediate state. This means also that attributes in MOF are considered to be fixed i.e one is sure that a property is in a certain state. To use distributions and thereby describe uncertainty that for example the attribute availability of a system is $95 \pm 2\%$ cannot be captured. The usefulness of continuous variables going along with the ability to describe distributions within the domain of EA analysis has been exemplified in [19] and [20]. In both publications distributions are used to describe modeled entities. These distributions then were used as input to perform EA analysis. In [19] the usage of *parametric dependencies* for performance prediction is suggested. On the other hand Närman et. al. [20] notes that reliability can be captured through the usage of continuous attributes. In order to support this kind of evaluation a power-full EA meta-language based on MOF needs to extent the attribute concept through an introduction of attributes that are capable to describe distributions and thereby express uncertainty (**Requirement R2**).

The third needed extension to MOF that was identified is the capability to express dependencies between attributes (**Requirement R3**). Attribute relationships are part of the Probabilistic Relational Models (PRM) formalism [21] (see explanation provided in 3.3), which has been proven to be a powerful mechanism to use for EA analysis. In [22] PRMs have been applied for (cyber) security analysis, whereas in [23] PRMs have been used to perform reliability evaluation. Thirdly in [24] PRMs are used to define patterns for quantitative EA analysis. The support of EA analysis using a formalism similar to PRM would facilitate the previously mentioned evaluations. This can be achieved for MOF through the introduction of relations between attributes (**Requirement R3.1**) and thus the modelling of their effects on each other (**Requirement R3.2**).

3 Related work

In this section we discuss two modelling approaches targeting enterprises and their architectures, respectively. Thereby, we do not focus on the concepts provided by the approaches, but put emphasis on the meta-languages that underlie these approaches. We discuss if and how these meta-languages address the requirements as stated in Section 2. In addition to these approaches, we also explain probabilistic relationships models as a promising means to incorporate uncertainty in domain models and to express relations between attributes.

3.1 ArchiMate (ORM)

The ArchiMate modelling language has a long history as an approach to model EAs. In the early work [25], the approach was based on an informally defined meta-language consisting of "things" and "relationships". In the most recent publications [26] this meta-language is replaced with the *Object-Role Model* (ORM) [27]. ORM supports basic modelling concepts grounded in the dichotomy between instance and corresponding entity type. Entity types can participate in n-ary relationships with each other and with value types, which are used to designate properties in an object-oriented sense. Additional facilities in ORM enable to model interrelationships between different relationships, e.g. describing that the set of values of one relationship, i.e. its tuples, is a subset of another relationship's values. In [4] Lankhorst et al. describe how architectural models based on ORM can be augmented with dependency information used for quantitative analyses of EAs. The modelling mechanisms are applied exemplarily, calling for an intuitive understanding of the underlying concepts. These concepts, which would contribute an extension to ORM, are contrariwise not discussed in detail. ArchiMate as presented in [4] does not use attributes, whereas modelling tools that support ArchiMate for example BiZZdesign Architect[3] allow to use profiles in order to describe classes. However this tool does not provide to model relations between attributes and is therefore not capable to fulfill the requirements as they have been stated in chapter 2.

3.2 MEMO Meta-Language

Frank discusses multi-perspective enterprise modelling (MEMO) [3] as an approach to integrate different perspectives on the enterprise. These perspectives are represented in different languages e.g. the *ITML* [28] modelling IT systems and resources. The languages are based on a common meta-language, the *MEMO metamodelling language* (MML) [29]. This language offers the syntactic primitives and model elements that are needed to describe EA conceptual models of the different special purpose languages. Basically, MML is an object-oriented metamodelling language introducing the meta-concepts MetaEntity, MetaAssociationLink, and MetaAttribute to designate classes, associations, and properties. Thereby, "classic" capabilities of object-oriented metamodelling facilities are provided, e.g. cardinalities or primitive data-types. A particular extensions is the *intrinsic* meta-concept. Frank furthers the discussion of Atkinson and Kühne in [30], who elaborate on the difficulties of describing different meta-levels within an object-oriented metamodel. Any modelling facility pursuing the paradigm of object-orientation centers around the dichotomy of meta-level, i.e. the level of the meta-concepts, and instance-level, on which the objects reside. Modelling domains that supply more than one level of *ontological instantiation* can hence not directly be covered by object-oriented modelling facilities. In order to avoid that language users introduce conceptually unclear 'workarounds' to the metamodels as the *type-item*-pattern, Frank adopts the idea of the *deep-instantiation.* He distinguishes the concepts along their *potency* in terms of Atkinson and Kühne [30]. A concept of potency n is instantiated to a concept of potency $n-1$, whereas concepts of potency 0 designate objects, instantiated associations, and bound attributes. Normal meta-concepts are of potency 1. Intrinsic meta-concepts are of potency 2, which are instantiated to normal meta-concepts.

The mechanism of the intrinsic concept is used by Frank et al. for the *Score-ML*, a language for defining (business) indicator systems within enterprise mod-

[3] http://www.bizzdesign.com/products/architect.html

els [31]. The Score-ML defines the concept of the SpecificIndicator that measures a ReferenceObject. Such reference object can be an arbitrary element in an metamodel underlying a particular enterprise modelling language, i.e. perspective on the enterprise. On the level of the indicator definition, the specific indicator can supply benchmarks for interpreting the particularValue of the indicator as measured at an instance of the reference object. Figure 1 shows the corresponding cutout from the metamodel of the Score-ML. Therein, we use the stereotype «i» to designate intrinsic meta-concepts.

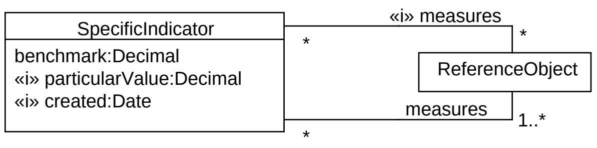

Fig. 1. Cutout of the indicator model of the Score-ML

In addition the Score-ML also supplies specialized relationship types that can be used to describe the dependencies between different indicators as well as between an indicator and a goal. One indicator can be computed from another indicator or can be similar to another one. These relationships are used during the design of the indicator system in different ways. Firstly, similar indicators can be exchanged with each other, in case one indicator is not measurable in the specific utilization context. Computational references are further operationalized during the design of an indicator system to actual computation rules that aggregate indicator values. Score-ML does not put particular emphasis on the notion of uncertainty, nor does it facilitate to introduce distributions acting as surrogates for actual indicator values.

3.3 Probabilistic Relational Models

As stated in the introduction of Section 3 PRMs serve as a guideline for the treatment of attributes, both in terms of support for uncertainty and relations between attributes. PRMs employ probabilistic reasoning to perform quantitative analysis of architecture properties. The main advantage of using PRMs instead of other probabilistic graphs, such as Bayesian networks [32] is their ability to also model objects of the world and their relations to each other in a manner similar to Entity-Relationship diagrams.

A PRM [21] specifies a template for a probability distribution over an architecture model. The template describes the metamodel for the architecture model, and the probabilistic dependencies between attributes of the architecture objects. A PRM, together with an instantiated architecture model of specific objects and relations, defines a probability distribution over the attributes of the objects. The probability distribution can be used to infer the values of unknown attributes, given the values of some attributes.

An architecture *metamodel* $\mathcal{M}$ describes a set of *classes*, $\mathcal{X} = X_1, \ldots, X_n$. Each class is equipped with a set of *descriptive attributes* and a set of *reference slots*. The set of descriptive attributes of a class X is denoted $\mathcal{A}(X)$. Attribute A of class X is denoted $X.A$ and its domain of *values* is denoted $V(X.A)$. For example, a class `Business Process` might have the attribute *Availability*, with domain $\{Up, Down\}$. The set of reference slots of a class X is denoted $\mathcal{R}(X)$. We use $X.\rho$ to denote the reference slot ρ of class X. Each reference slot ρ is typed with the *domain type* $Dom[\rho] = X$ and the *range type* $Range[\rho] = Y$, where $X; Y \in \mathcal{X}$. A slot ρ denotes a relation from X to Y. Slots are similar to the relations in Entity-Relationship diagrams. For example we might have a class `Business Process` with the reference slot *uses* whose range is the class `Business Application`. For each reference slot ρ we have an inverse reference slot ρ^{-1} denoting the inverse relation. In the prior example, the class `Business Application` has an inverse reference slot $uses^{-1}$ to `Business Process`.

An architecture *instantiation* $\mathcal{I}$ (or an architecture model) specifies the set of objects of each class, the values for the attributes, and the references of the objects. It specifies a particular set of data objects, business processes, business applications, etc., along with their attribute values and references. We also define a *relational skeleton* σ_r as a partial instantiation which specifies the set of objects in all classes as well as all the reference slot values, but not the attribute values.

A probabilistic relational model Π specifies a probability distribution over all instantiations $\mathcal{I}$ of the metamodel $\mathcal{M}$. This probability distribution is specified similar to a Bayesian network [32], which consists of a qualitative dependency structure and associated quantitative parameters.

The *qualitative* dependency structure is defined by associating each attribute $X.A$ a set of parents $Pa(X.A)$ through attribute relations. Each parent of $X.A$ is defined as $X.\tau.B$ where $B \in \mathcal{A}(X.\tau)$ and τ is either empty, a single slot ρ or a sequence of slots $\rho_1, \ldots, \rho_k$ such that for all i, $Range[\rho_i] = Dom[\rho_{i+1}]$. In our example, the attribute *BusinessProcess.Availability* may have *BusinessApplication.Availability* (i,e. $Pa(BusinessApplication.Availability) = BusinessProcess.Uses^{-1}.Availability$) as parent, thus indicating that the Availability of an `Business Process` depends on the availability performed by the `Business Application`, which realize the `Business Process`. Note that $X.\tau.B$ may reference a set of attributes rather than a single one. In these cases, we let A depend probabilistically on some aggregated property over those attributes, such as the logical operations MIN or MAX. Considering the *quantitative* part of PRMs, given a set of parents for an attribute, we can define a local probability model by associating a *conditional probability distribution* (CPD) with the attribute, $P(X.A|Pa(X.A))$. We can now define a PRM Π for a metamodel $\mathcal{M}$ as follows. For each class $X \in \mathcal{X}$ and each descriptive attribute $A \in \mathcal{A}(X)$, we have a set of *parents* $Pa(X.A)$, and a CPD that represents $P_\Pi(X.A|Pa(X.A))$. Given a *relational skeleton* σ_r (i.e. a metamodel instantiated to all but the attribute values), a PRM Π specifies a probability distribution over a set of instantiations $\mathcal{I}$ consistent with σ_r:

$$P(\mathcal{I}|\sigma_r, \Pi) = \prod_{x \in \sigma_r(X)} \prod_{A \in \mathcal{A}(x)} P(x.A|Pa(x.A))$$

where $\sigma_r(X)$ are the objects of each class as specified by the relational skeleton σ_r. A PRM thus constitutes the formal machinery for calculating the probabilities of various architecture instantiations. This allows us to infer the probability that a certain attribute assumes a specific value, given evidence of the rest of the architecture instantiation.

4 Extending the Meta Object Facility

The meta-object facility (MOF) [33] is an OMG standard that provides a multi-purpose metamodel for defining object-oriented models based on the UML infrastructure [34][4]. According to the findings in [16], MOF is prevalent as metamodel for defining EA information models with the exceptions as discussed in Section 3. With respect to requirements **R1**-**R3** MOF does not offer dedicated modelling mechanisms, neither for modelling existential uncertainty, uncertainty concerning values, nor dependencies between architectural properties. Below we introduce an extension to MOF that addresses the requirements. This extension builds on EMOF that defines basic concepts for object-oriented metamodels. Following Figure 2 displays the stack of meta-levels on which our modelling approach builds and indicates on which meta-level our extended EMOF resides.

Requirement **R1.1** is addressed by a model extension that allows to designate an existence probability for an instance of a particular class, i.e. for an InstanceSpecification in terms of MOF. An additional attribute exProbability can be used to describe that an instance has a lower existence probability than the default value 1. This value is retained to ensure compatibility with 'normal', i.e. non-extended, MOF models. The existential uncertainty with respect to relationships between instances as well as the uncertainty with respect to the value of a property are addressed in a uniform manner. This abides to the fact that in EMOF, the concept property is used to describe both simple properties but also unidirectional relationships. The reference property.opposite is used to designate that two such unidirectional relationships constitute a bi-directional relationship in the sense of conceptual modelling. The value of a property at a particular instance is reflected in a slot which in turn is assigned a valueSpecification containing the actual value. Different types of value specifications are supplied by MOF, of which three are of particular interest for our extension. IntegerExpression and FloatExpression are used to specify values for Integer and Float properties, reflectively. We introduce these types of expressions as specializations to the generic ValueExpression of MOF that allows to specify untyped values. Both specialized expressions contain an attribute uncertainty that allows to specify, the degree of

[4] MOF can also be used to 'boot-strap' itself, i.e. the concepts of MOF can be explained as instances of MOF concepts.

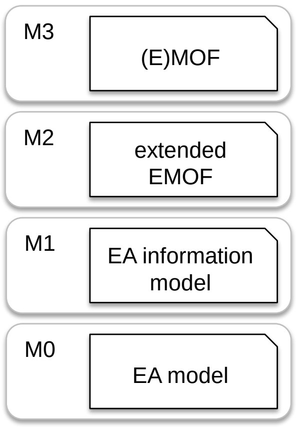

Fig. 2. Meta-modelling stack

uncertainty in terms of the *variance*. We thereto assume two specific distributions for discrete and continuous properties, namely the binomial distribution and the normal distribution, correspondingly. The InstanceValue designates the instance specification assigned to a non-primitive, i.e. referencing, property. It further assigns a probability for the corresponding value assignment being correct. Using these mechanisms, we address requirements **R1.2** and **R2**, respectively. The following constraint is needed to ensure that properties representing bi-directional relationships are consistent with respect to their assigned probabilities:

```
context: InstanceValue
inv: InstanceValue->forall(i|self.owningSlot.definingFeature.opposite ==
i.owningSlot.definingFeature implies
self.proability == i.probability);
```

We fulfill **R3.1** through the introduction of the PropertyRelationship concept. A PropertyRelationship is a directed relationship between two properties that specifies influence from the source to the dependent property. The two properties may be owned by the same class or by transitively associated ones. In the latter case, the property relationship further specifies, which relationship path between the classes is influencing the dependency from the source to the target property. This means that the following constraints have to hold:

```
context: PropertyRelationship
inv: targetPath()->forAll(c|isSelfOrSub(c,sPath().at(tPath().indexOf(c))))
inv: sourceProperty.type.oclIsType(DataType)
inv: dependentProperty.type.oclIsType(DataType)
inv: dependencyPath->forAll(c|dependencyPath->count(c) == 1)
inv: sourceProperty <> dependentProperty
```

In these constraints we build on auxiliary operations:

```
context: AttributeRelationship
```

```
allSuperClasses(c:Class):Collection
allSuperClasses = c.superClass->collect(c|allSuperClasses(c)).flatten()
context: PropertyRelationship
isSelfOrSub(Class sub, Class super):Boolean
isSelfOrSub = (sub == super) or allSuperClasses(sub).contains(super)
context: PropertyRelationship
sPath():Sequence
sPath = dependencyPath->collect(p|p.type).prepend(sourceProperty.class)
context: PropertyRelationship
tPath():Sequence
tPath = dependencyPath->collect(p|p.class).append(dependentProperty.class)
```

Finally requirement **R3.2** is addressed through a link between PropertyRelationship and OpaqueExpression. The standard [13] defines an opaque expression as "an uninterpreted textual statement that denotes a (...) set of values when evaluated in a context". Using such expressions, we establish definitory dependencies between different properties, i.e. designate that and how one property can be computed from another.

Figure 3 shows the extended metamodel and highlights added or adapted meta-classes.

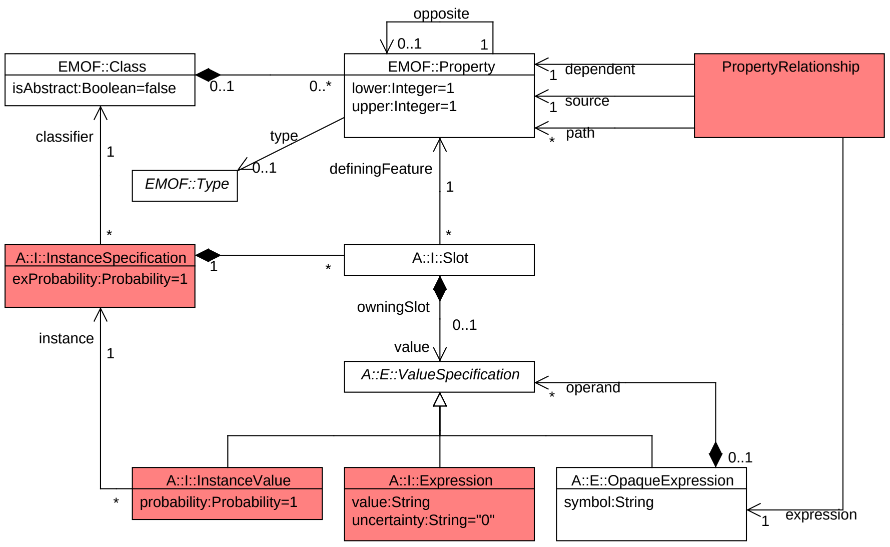

Fig. 3. Metamodel of the EMOF extension

5 Exemplifying the language for EA analysis

The basis for EA analysis are architecture models, which provide abstractions from the real world concepts. Dependencies between the modeled concepts play, as alluded to above, an important role in EA documentation and analysis. The subsequent example is concerned with metrics for application landscapes, which are an important part of the overall EA. The presented model, similar to the model introduced by Lankes in [35], discusses the availability of services offered by a business application in respect to the services used by the business application. Thereby, the aspect of failure propagation in the application landscape can be modeled on an abstract level.

The classes, attributes, and associations introduced therein are defined as follows:

BusinessApplication refers to a system, which is implemented in software, deployed at a specific location, and which provides support for at least one of the company's business processes. As consequence of the (not modeled) dependency of a business application to an underlying hardware device, the application has a certain level of availability (modeled as the probability for being available). In performing the business support, a business application may depend on other applications, which is modeled by two associations to the offered or used interfaces. The availability of the business application is thus dependent on the availability of the underlying hardware (not modeled) as well as on the availability of the interfaces used.

Interface is offered by a business application to provide a service for external use by one or more other applications or business processes. As a consequence of the provision by an application, the interface has an availability associated to the availability of the offering application.

BusinessProcess refers to a sequence of individual functions with connections between them. A business process as used in this model should not be identified with a single process step, but with high-level processes at a level similar to the one used in value chains. The execution of the process is dependent on the availability of the applications used.

Figure 4 shows the interplay between these concepts.

The dependencies between the values of the different availability attributes of the business application, the interface, and the business process class, which are informally defined in the textual descriptions, are visually indicated using the aforementioned notation for property relationships. In addition, the dependency between the availability of an interface and of the offering business application's availability can be expressed in simple terms, i.e. they are equal. A value specification can express this type of relationship. Figure 5 shows an instantiation of the information model and calculates the probabilities for the availability given the corresponding uncertainties. Thereby, especially the uncertainty with respect to the interface LOG is taken into account. According to the current knowledge, this interface exists with a probability of 50% (exProbability = 0.5).

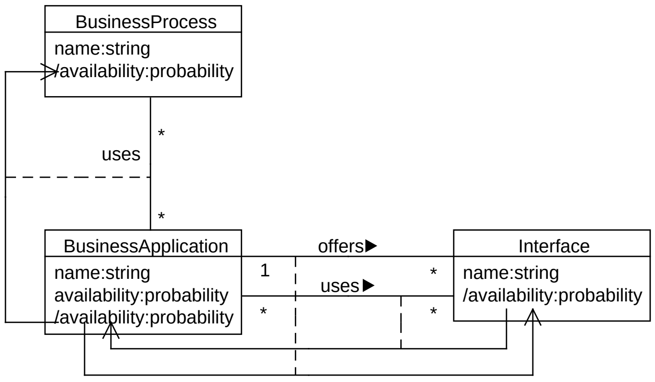

Fig. 4. Information model of the example

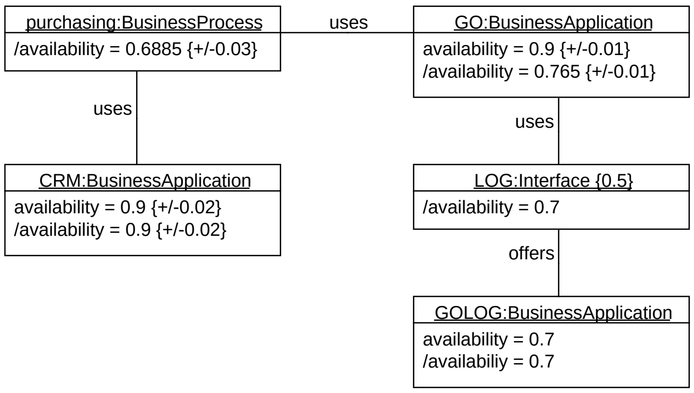
purchasing:BusinessProcess
/availability = 0.6885 {+/-0.03}
uses
GO:BusinessApplication
availability = 0.9 {+/-0.01}
/availability = 0.765 {+/-0.01}
uses
CRM:BusinessApplication
availability = 0.9 {+/-0.02}
/availability = 0.9 {+/-0.02}
uses
LOG:Interface {0.5}
/availability = 0.7
offers
GOLOG:BusinessApplication
availability = 0.7
/availabiliy = 0.7

Fig. 5. Instance model of the example

6 Outlook

During the description of current states of the EA and the development of future states of the EA (cf. [36] for typical activities of EA management) two different types of uncertainty have to be considered: uncertainty with respect to the described current state, i.e. is the gathered information correct, or uncertainty with respect to a future state, i.e. is the plan to come true or did we make the right assumptions with respect to the future evolution. Are the right projects selected to come true. Combinations of the aforementioned uncertainties are also possible. In the approach only the describe uncertainty is discussed.

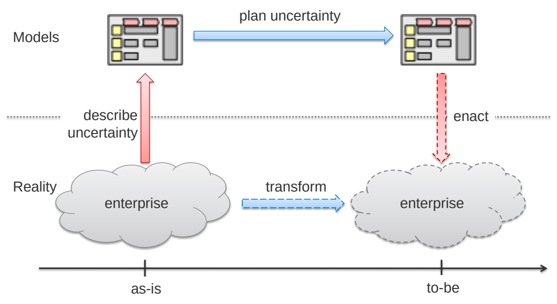

Fig. 6. Different types of uncertainty

Another interesting direction for future work targets the exact nature of the relationships between properties. The current model can describe that a relationship exists and can detail this relationship with an expression for their computation. On a more fine-grained level, we could distinguish between definitory dependencies and causal dependencies, of which the former are used to define a property, whereas the latter describe that a property depends on the value of another one. In both cases, it would also be possible to specify the direction of the dependency, i.e. to designate, whether dependent and depending property behave opposite or similar. While such specification is less expressive than the an actual computable expression, it can be used for qualitative analyses on the dependencies and the mutual reaction on changes.

Finally, the presented approach would benefit from an implementation. Therefore, existing tools (cf. [1]) can be extended to incorporate the MOF extension as presented in this paper. The implementation would further enable the evaluation of the meta-language in practice.

References

1. Buschle, M., Ullberg, J., Franke, U., Lagerström, R., Sommestad, T.: A tool for enterprise architecture analysis using the PRM formalism. In: CAiSE2010 Forum PostProceedings. (2010)
2. Buckl, S., Ernst, A.M., Lankes, J., Matthes, F.: Enterprise Architecture Management Pattern Catalog (Version 1.0, February 2008). Technical report, Chair for Informatics 19 (sebis), Technische Universität München, Munich, Germany (2008)
3. Frank, U.: Multi-perspective enterprise modeling (memo) – conceptual framework and modeling languages. In: 35th Hawaii International Conference on System Sciences (HICSS 2002), Washington, DC, USA (2002) 1258–1267
4. Lankhorst, M.M.: Enterprise Architecture at Work: Modelling, Communication and Analysis. 2nd edn. Springer, Berlin, Heidelberg, Germany (2009)
5. Dern, G.: Management von IT-Architekturen (Edition CIO). Vieweg, Wiesbaden, Germany (2006)
6. Niemann, K.D.: From Enterprise Architecture to IT Governance – Elements of Effective IT Management. Vieweg+Teubner, Wiesbaden, Germany (2006)

7. The Open Group: TOGAF "Enterprise Edition" Version 9. `http://www.togaf.org` (cited 2010-02-25) (2009)
8. Department of Defense (DoD) USA: DoD Architecture Framework Version 2.0: Volume 1: Introduction, Overview, and Concepts – Manager's Guide. `http://www.defenselink.mil/cio-nii/docs/DoDAF\%20V2\%20-\%20Volume\%201.pdf` (cited 2010-02-25) (2009)
9. Matthes, F., Buckl, S., Leitel, J., Schweda, C.M.: Enterprise Architecture Management Tool Survey 2008. Chair for Informatics 19 (sebis), Technische Universität München, Munich, Germany (2008)
10. Murer, S., Worms, C., Furrer, F.J.: Managed evolution. Informatik Spektrum **31**(6) (2008) 537–547
11. Buckl, S., Ernst, A.M., Lankes, J., Schneider, K., Schweda, C.M.: A pattern based approach for constructing enterprise architecture management information models. In Oberweis, A., Weinhardt, C., Gimpel, H., Koschmider, A., Pankratius, V., Schnizler, eds.: Wirtschaftsinformatik 2007, Karlsruhe, Germany, Universitätsverlag Karlsruhe (2007) 145–162
12. Kurpjuweit, S., Winter, R.: Viewpoint-based meta model engineering. In Reichert, M., Strecker, S., Turowski, K., eds.: 2^{nd} International Workshop on Enterprise Modelling and Information Systems Architectures (EMISA 2007). LNI, Bonn, Germany, Gesellschaft für Informatik (2007) 143–161
13. Object Management Group (OMG): Omg unified modeling language^{TM}(omg uml), superstructure – version 2.3 (formal/2010-05-05) (2010)
14. Object Management Group (OMG): Meta object facility (mof) core specification, version 2.0 (formal/06-01-01) (2006)
15. Buckl, S., Matthes, F., Schweda, C.M.: A meta-language for ea information modeling – state-of-the-art and requirements elicitation. In Bider, I., Halpin, T., Krogstie, J., Nurcan, S., Proper, E., Schmidt, R., Ukor, R., eds.: Enterprise, Business-Process and Information Systems Modeling. Lecture Notes in Business Information Systems, Springer (2010) 169–181
16. Buckl, S., Matthes, F., Schweda, C.: A meta-language for EA information modeling–state-of-the-art and requirements elicitation. Enterprise, Business-Process and Information Systems Modeling (2010) 169–181
17. Johnson, P., Lagerström, R., Närman, P., Simonsson, M.: Enterprise architecture analysis with extended influence diagrams. Information Systems Frontiers **9**(2) (2007) 163–180
18. Aier, S., Buckl, S., Franke, U., Gleichauf, B., Johnson, P., Närman, P., Schweda, C., Ullberg, J.: A survival analysis of application life spans based on enterprise architecture models. In: 3rd International Workshop on Enterprise Modelling and Information Systems Architectures, Ulm, Germany. (2009) 141–154
19. Becker, S., Koziolek, H., Reussner, R.: The Palladio component model for model-driven performance prediction. Journal of Systems and Software **82**(1) (2009) 3–22
20. Närman, P., Buschle, M., König, J., Johnson, P.: Hybrid Probabilistic Relational Models for System Quality Analysis. In: Enterprise Distributed Object Computing Conference (EDOC), 2010 14th IEEE International, IEEE 57–66
21. Friedman, N., Getoor, L., Koller, D., Pfeffer, A.: Learning probabilistic relational models. In: International Joint Conference on Artificial Intelligence: Stockholm, Sweden. Volume 16., Citeseer (1999) 1300–1309
22. Sommestad, T., Ekstedt, M., Johnson, P.: A probabilistic relational model for security risk analysis. Computers & Security **29**(6) (September 2010) 659–679

23. König, J., Nordstrom, L., Ekstedt, M.: Probabilistic Relational Models for assessment of reliability of active distribution management systems. In: Probabilistic Methods Applied to Power Systems (PMAPS), 2010 IEEE 11th International Conference on, IEEE (2010) 454–459
24. Buckl, S., Franke, U., Holschke, O., Matthes, F., Schweda, C., Sommestad, T., Ullberg, J.: A pattern-based approach to quantitative enterprise architecture analysis. In: 15th Americas Conference on Information Systems (AMCIS). (2009)
25. Lankhorst, M.M.: Introduction to enterprise architecture. In Lankhorst, M., ed.: Enterprise Architecture at Work, Berlin, Heidelberg, New York, Springer (2005)
26. Jonkers, H., van den Berg, H., Iacob, M.E., Quartel, D.: Archimate extension for modeling togaf's implementation and migration phases. `http://www.bizzdesign.com/index.php/component/docman/doc_download/18-archimate-extension-for-modeling-togafs-implementation-and-migration-phases` (2010)
27. Halpin, T.: Object-role modeling (ORM/NIAM). Handbook on Architectures of Information Systems (2006) 81–103
28. Kirchner, L.: Eine Methode zur Unterstützung des IT-Managements im Rahmen der Unternehmensmodellierung. PhD thesis, Universität Duisburg-Essen, Berlin, Germany (2008)
29. Frank, U.: The memo meta modelling language (mml) and language architecture (icb-research report). Technical report, Institut für Informatik und Wirtschaftsinformatik, Duisburg-Essen, Germany (2009)
30. Atkinson, C., Kühne, T.: Reducing accidental complexity in domain models. Software and Systems Modeling (2007) 345–359
31. Frank, U., Heise, D., Kattenstroth, H., Schauer, H.: 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)
32. Jensen, F., Nielsen, T.: Bayesian networks and decision graphs. Springer Verlag, New York (2007)
33. Object Management Group (OMG): Meta object facility (mof) specification, version 2.0 (2006)
34. Object Management Group (OMG): Omg unified modeling language$^{\text{TM}}$(omg uml), infrastructure – version 2.3 (formal/2010-05-03) (2010)
35. Lankes, J.: Metrics for Application Landscapes – Status Quo, Development, and a Case Study. PhD thesis, Technische Universität München, Fakultät für Informatik, Munich, Germany (2008)
36. Buckl, S., Matthes, F., Schweda, C.M.: Towards a method framework for enterprise architecture management – a literature analysis from a viable system perspective. In: 5$^{\text{th}}$ International Workshop on Business/IT Alignment and Interoperability (BUSITAL 2010). (2010)