An Extension to the Essential Meta-Object Facility (EMOF) for Specifying and Indicating Dependencies between Properties

Technical Report TB 0801

Sabine M. Buckl, Alexander M. Ernst, Christian M. Schweda

Software Engineering for Business Information Systems (sebis)
Ernst Denert-Stiftungslehrstuhl
Chair for Informatics 19
Technische Universität München

Boltzmannstraße 3, 85748 Garching b. München, Germany

{sabine.buckl | ernst | schweda}@in.tum.de

November 2008

1 Scope and motivation

Models are a useful technique in many different disciplines ranging from data maps in geography [Tu01] via goal and value modeling in requirements engineering [Go06] to class diagrams in software engineering [Hi05]. Thereby, models are used to reduce complexity by abstracting from the real world and focusing on properties relevant to answer the problems, which should be addressed by the model [St73]. A city map used for route searching for example does not account for every traffic light within the city, but brings into focus the streets and their intersections.

The models described above share the communality that they are implicitly or explicitly based on metamodels. Using the concepts provided by a metamodeling facility, as the Meta Object Facility (MOF) [OM06a], a model can capture especially structured aspects of the modeled object.

Nevertheless, these facilities are not directly capable of expressing all types of dependencies between the modeled concepts. This is especially true, if mechanisms for deriving values are considered. While languages, like the Object Constraint Language (OCL) [OM06b], can be used to specify rules for deriving values, a more abstract notion of value dependency is not provided as part of the MOF.

Such a notion would nevertheless be beneficiary in the context of certain domain-specific object-oriented modeling languages to express **that some kind** of dependency between attributes exists, while not having to indicate, of **what kind** this dependency is. An initial model containing abstract dependencies could then be refined to employ computable dependency expressions in an appropriate language, e.g. OCL.

The extension to the MOF, more precise to the Essential MOF (EMOF), presented below, is based on the work of [Jo07] and targets to close the aforementioned gap in abstract dependency modeling 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.

2 Conformance to modeling standards

The extension provided in this document is compliant to the modeling facility as specified in the Essential MOF. The extension further makes use of modeling concepts introduced in the UML Infrastructure Specification [OMG05a], more precisely from the *core::relationship package*. The EMOF was chosen as a basis for this approach to provide a minimum starting point for modeling – abstaining from the more complex and sophisticated concepts of the Complete MOF (CMOF). Nevertheless, the concepts as presented below should also be usable in the context of CMOF.

The remainder of this report is structured as follows: the metamodel of the extension is introduced starting at the level of package dependencies and is subsequently refined via class level concepts, displayed using the graphical notation of UML [OM05b]. Finally, a detailed description of the introduced classes concludes the specification part of this report further comprising a set of constraints, which apply, if the proposed metamodel is used. In order to exemplify the proposed extension, an application example is given in Section 4, which complements the paper.

3 Specification of the metamodel extension: AttributeRelationship

The following metamodel (see Figure 1) introduces the core concept of the proposed extension, the attribute relationship.

Description

An attribute relationship is a directed relationship between two properties that specifies influence from the source to the dependent (target) property. The two properties may be owned by the same class or by transitively associated ones. In the latter case, the attribute relationship further specifies, which relationship path between the classes is influencing the dependency from the source to the target property.

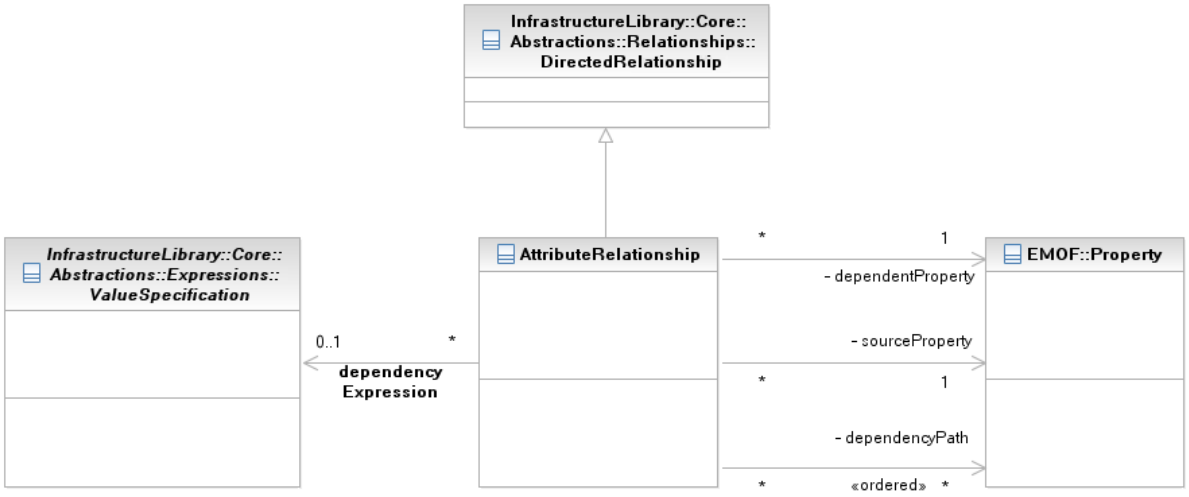

Figure 1 – Metamodel defining the attribute relationship concept

Generalizations

- "DirectedRelationship" as introduced in the UML Infrastructure Specification [OM05a] in the Package *Core::Abstractions::Relationships*.

Attributes

No additional attributes.

Associations

- `sourceProperty: Property[1]` – Specifies the property, from which influence is exerting in this relationship.
- `dependentProperty: Property[1]` – Specifies the property, of which the value is influenced in this relationship.
- `dependencyPath: Property[0..*]` – Specifies the path from the class holding the source property to the class holding the target property via which the influence is "transported".
- `dependencyExpression: ValueSpecification[0..1]` – Specifies the computation rule delivering the respective value, if evaluated, for the dependent property.

Constraints

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

Additional Operations

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

```
context AttributeRelationship
      targetPath():Sequence
      targetPath = dependencyPath->collect(p|p.class)
                                          .append(dependentProperty.class)
```

Notation

An attribute relationship may be drawn as a solid line connecting two properties (denoted textually), having an arrow on the dependent property end (see Figure 2). Additionally, the solid line has additional dashed lines attached to it (cf. Figure 3), to reference the properties from the dependency path[1]. No way for annotating the ordering of the dependency path properties is provided, as the ordering should be derivable unambiguously.

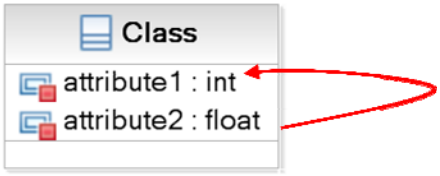

Figure 2 – Notation for an attribute relationship in one class

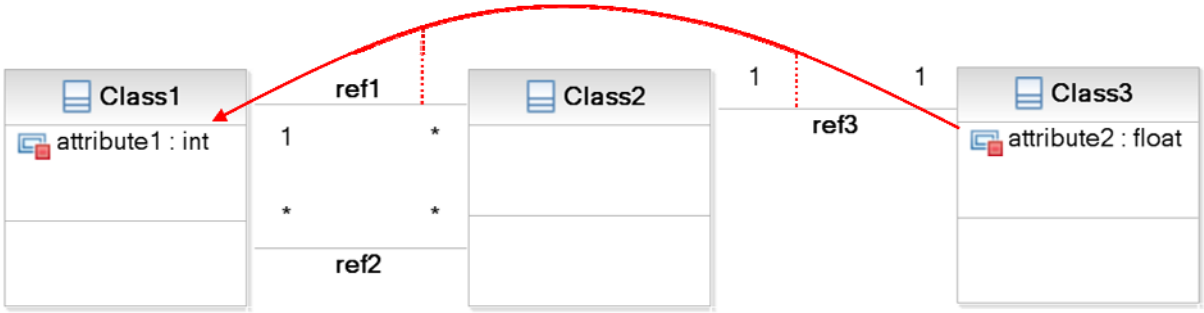

Figure 3 – Notation for a traversing attribute relationship

4 Application example

Within the area of enterprise architecture (EA) management, models form an important means to deal with the rising complexity originating in this context. Various approaches to deal with this complexity exist, emerging from different sources as e.g. academia [FAW07, Fr02, La05a, Wi07], practitioners [De06, En08, Ke07, TO02], and public authorities [Do04a, Do04b]. Although these approaches differ widely regarding the extent to which business and IT aspects are taken into consideration, they share the common idea that not solely the artifacts, e.g. business applications, business processes, projects, strategies, are of importance but additionally also the dependencies between them.

The basis for analysis operations on EAs are documentations of these architectures, which are widely created using object-oriented models [Br05, La05b]. These models are used to provide abstractions for the respective real world concepts. Dependencies between the modeled concepts play, as alluded to above, an important role in EA documentation and analysis. Therefore, this area can be used to draw an application example for modeling attribute relationships from. This is especially true, as the knowledge of a dependency between different attributes is more likely to be present to the enterprise architects than the exact type of dependency.

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 in [La08], 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 a highly abstract level. Figure 4 introduces the annotated conceptual model for the application example.

[1] The properties may therein be also denoted as solid lines indicating the respective associations between the classes.

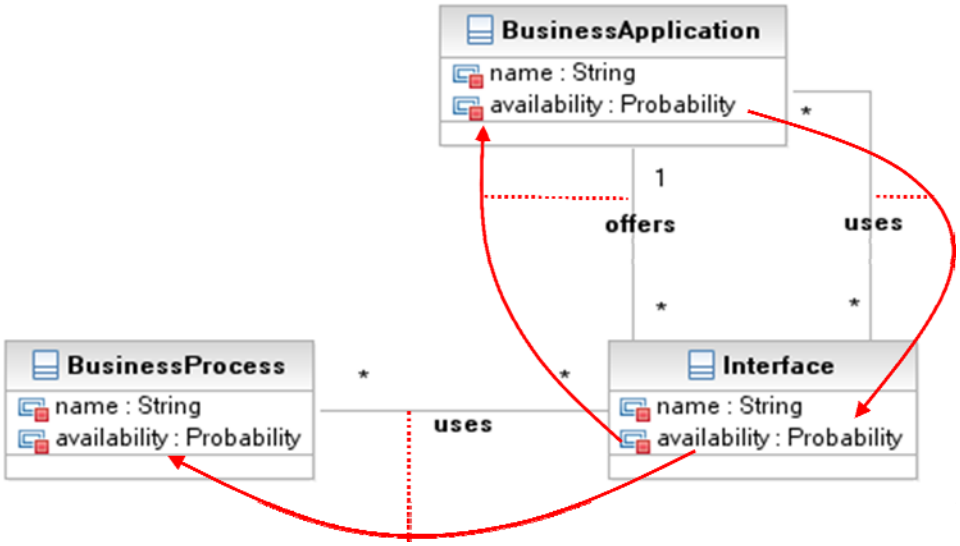

Figure 4 – A notated class model of the application example

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 interfaces `used`. Thus, the process has an individual `availability` assigned to it.

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 attribute 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 respective value specification can express this type of relationship. Concerning the availability of a using business process, a value specification cannot be found that easy, as e.g. considerations on the interfaces failing independently would have to be undertaken.

References

[Br05] Brendebach, K.: *Integrierte Modelle uns Sichten für das IT-Management – Analyse und Entwicklung in Zusammenarbeit mit der HVB Systems*. Diploma thesis, Technische Universität München, 2005 (in German).

[De06] Dern, G.: *Management von IT-Architekturen (Edition CIO).* Vieweg, Wiesbaden, 2006 (in German).

[Do04a] Department of Defence Architecture Framework Working Gorup: *DOD Architecture Framework Version 1.0, Volume I: Definitions and Guidelines.* USA, 2004.

[Do04b] Department of Defence Architecture Framework Working Gorup: *DOD Architecture Framework Version 1.0, Volume II: Product Descriptions.* USA, 2004.

[En08] Engels, G., Hess, A.,Humm, B., Juwig, O., Lohmann, M., Richter, J.-P., Voss, M., Willkomm, J.: *Quasar Enterprise – Anwendungslandschaften serviceorientiert gestalten.* dpunkt.verlag, Heidelberg, 2008 (in German).

[FAW07] Fischer, R., Aier, S., Winter, R.: *A Federated Approach to Enterprise Architecture Model Maintenance.* In: *Enterprise Modelling and Information System Architectures Proceedings of the 2nd International Workshop EMISA 2007, pages 9-22, St. Goar, Rhine, 2007.*

[Fr02] Frank, U.:*Multi-Perspective Enterprise Modeling (MEMO) – Conceptual Framework and Modeling Languages*. In: *Proceedings of the 35th Annual Hawaii International Conference on System Sciences 35,* Honolulu, 2002.

[Go06] Gordijn, J.; Petit, M.; Wieringa, R.: *Understanding Business Strategies of Networked Value Constellations Using Goal- and Value Modeling*. In: Proceedings the 14th IEEE International Conference on Requirements Engineering, Minneapolis, 2006.

[Hi05] Hitz, M.; Kappel, G.; Kapsammer, E.; Retschitzegger, W.: *UML@Work*. 3. Auflage, 2005. ISBN 3898642615 (in German).

[Jo07] Johnson, P., Johansson, E., Sommestad, T. And Ullberg, J.: *A Tool for Enterprise Architecture Analysis*. In: *11th IEEE International Enterprise Distributed Object Computing Conference*. Annapolis, Maryland, 2007.

[Ke07] Keller, W.: *IT- Unternehmensarchitektur*. dpunkt.verlag, 2007 (in German).

[La05a] Lankhorst, M.: *Enterprise Architecture at Work*. Springer, Berlin, Heidelberg, 2005.

[La05b] Lauschke, S.: *Softwarekartographie: Analyse und Darstellung der IT-Landschaft eines mittelständischen Unternehmens*. BSc thesis, Technische Universität München, 2005 (in German).

[La08] Lankes, J.: *Metrics for Application Landscapes. Status Quo, Development, and a Case Study*. Phd thesis, Fakultät für Informatik, Technische Universität München, München, 2008

[TO02] The Open Group: *TOGAF "Enterprise Edition" Version 8.1*. The Open Group, 2002. http://www.opengroup.org/architecture/togaf8-doc/arch/ (cited 2008-10-24).

[OM06a] OMG: *Meta Object Facility (MOF) Core Specification, version 2.0 (formal/06-01-01).* Object Management Group, 2006.

[OM06b] OMG: *OCL 2.0 Specification (formal/2006-05-01).* Object Management Group, 2006.

[OM05a] OMG: *UML 2.0 Infrastructure (formal/2005-07-05).* Object Management Group, 2005.

[OM05b] OMG: *UML 2.0 Superstructure (formal/2005-07-04).* Object Management Group, 2005.

[St73] Stachowiak, H.: *Allgemeine Modelltheorie*. Springer-Verlag, Wien, 1973. ISBN 3211811060 (in German).

[Tu01] Tufte, E. R.: *The Visual Display of Quantative Information*. Second Edition, Graphics Press LLC, 2001.

[Wi07] Wittenburg, A.: *Softwarekartographie: Modelle und Methoden zur systematischen Visualisierung von Anwendungslandschaften.* Phd thesis, Fakultät für Informatik, Technische Universität München, München, 2007 (in German).