Modeling the Linguistic Architecture of Software Products

Jean-Marie Favre[1], Ralf Lämmel[2], and Andrei Varanovich[2]

[1] Université Joseph Fourier, Grenoble, France
[2] Software Languages Team, Universität Koblenz-Landau, Germany

Abstract. Understanding modern software products is challenging along several dimensions. In the past, much attention has been focused on the logical and physical architecture of the products in terms of the relevant components, features, files, and tools. In contrast, in this paper, we focus on the linguistic architecture of software products in terms of the involved software languages and related technologies, and technological spaces with linguistic relationships such as membership, subset, or conformance. We develop a designated form of megamodeling with corresponding language and tool support. An important capability of the megamodeling approach is that entities and relationships of the megamodel are linked to illustrative software artifacts. This is particularly important during the understanding process for validation purposes. We demonstrate such megamodeling for a technology for Object/XML mapping. This work contributes to the *101companies* community project.

Keywords: Megamodel. Linguistic architecture. Software language. Software technology. Technological space. Object/XML mapping. MegaL.

1 Introduction

Understanding modern software products is challenging because they are complex along several dimensions. We are specifically interested in the complexity of software products due to the involved *software languages*, *software technologies*, and *technological spaces* [7, 17] while possibly leveraging generative, reflective, transformational, and model-driven engineering approaches.

Consider, for example, web applications. A given application may leverage several software languages simultaneously: programming languages (e.g., Java, PHP, JavaScript, or Python), reusable domain-specific languages (e.g., CSS, XSLT, or SQL), library-based languages (e.g., JQuery, DOM), implicit languages relying on particular frameworks or annotation schemas (e.g., particular configuration or mapping languages for frameworks such as Hibernate or JAXB) as well as problem-specific languages (such as the object models, schemas, or domain-specific languages specifically designed for the application). Artifacts of these languages are processed, generated, or affected by compilers, interpreters, code generators, annotation injectors, etc.

In order to facilitate the understanding of current software languages and software technologies, the *101companies* community project [11][3] aims at devel-

[3] http://101companies.org

oping a free, structured, web-accessible knowledge resource including an open-source repository for different stakeholders with interests in software technologies, software languages, and technological spaces; notably: teachers and learners in software engineering or software languages as well as software developers, software technologists, and ontologists.

In the context of the *101companies* project, the present paper[4], introduces a *megamodeling* approach supported by a language MegaL and associated tools for editing and exploration of megamodels. According to [1]: "A megamodel is a model of which at least some elements represent and/or refer to models or metamodels." Different forms of megamodeling have been introduced and utilized elsewhere [2,3,14,21,26,27]. Existing work essentially focuses on modeling typical Model Driven Engineering entities such as models, metamodels and transformations. Instead, we focus on conceptual entities such as (software) languages and (software) technologies as well as a range of so-called digital entities including languages, language processors, programs, libraries, files, directories, source files, meanings of programs, and in-memory structures such as object graphs. For instance, only by including languages (as opposed to their description by metamodels or grammars), we are able to explain certain linguistically founded aspects of software technologies.

We also say that the developed megamodeling approach addresses the 'linguistic architecture' of software products, thereby complementing other, more established dimensions of architecture: the 'logical architecture' as it is the subject of 'classical' software architecture as well as specific paradigms such as component-, feature- or aspect-oriented software development; the 'physical architecture' which is typically concerned with building, packaging, and deploying software and hence, with entities such as files and servers.

Contributions of the paper

- ◇ We develop a megamodeling approach that is useful for understanding the linguistic architecture of software products in terms of the involved languages, technologies, and linguistic relationships. This approach is supported by the MegaL language and an associated tool suite under development.
- ◇ We demonstrate megamodeling in the challenging context of Object/Relational/XML [19,22,29] mapping (or O/R/X mapping). In the paper, we focus on one *O/X mapping* technology with a megamodel that is highly abstract but still it includes the key characteristics of the technology in question.
- ◇ The value of our megamodels is a cognitive one: they facilitate understanding of technologies and usage thereof. We strongly improve such cognitive value by enabling a form of *linked megamodels* such that entities and relationships are linked to resources (e.g., in the *101companies* repository) so that megamodels can be explored and validated.

[4] The paper's website `http://softlang.uni-koblenz.de/mega/` provides supplementary material including some megamodels of software products and technologies.

Non-contributions of the paper Note that the goal of this paper is by no means to define the ultimate megamodeling language in this context (such as MegaL) in any exhaustive or formal way, but just on the contrary, to motivate the overall approach and to illustrate its value with a concrete example. Also, the megamodeling approach, as it stands, does not yet readily support any sort of automated analysis or verification of software products. Instead, the current focus is on enabling the description of entities and relationships in a linguistic architecture in a way that is fit for validation purposes during the human understanding process.

Road-map of the paper §2 motivates and illustrates the notion of linguistic architecture. §3 describes entity and relationship types needed for modeling linguistic architecture. §4 develops an initial megamodel for O/X mapping that abstracts essentially from most aspects of the concrete O/X mapping technology and the concrete software that uses O/X mapping. §5 derives a more detailed megamodel that also captures interesting .NET-specific characteristics of O/X mapping. §6 develops the notion of linked megamodels. §7 discusses related work. §8 concludes the paper.

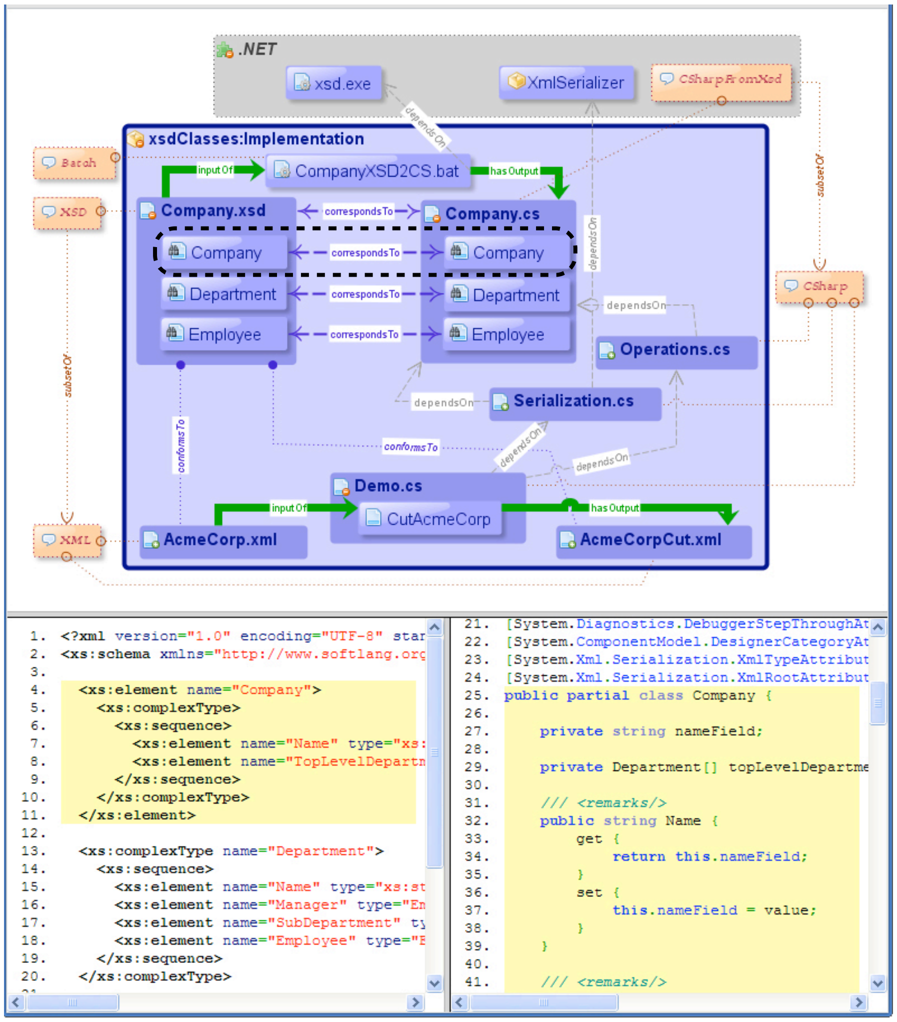
2 Illustration of linguistic architecture

Consider the upper frame in Figure 1. (The lower frame will be discussed in §6.) The linguistic architecture of a software product is described in the MegaL/yEd visual notation.[5] The product is a *C#*-based application which makes use of .NET's Object/XML mapping technology.[6] In fact, the product is a *101companies* implementation, which is named *xsdClasses* and available online. Hence, the application deals with companies (as in the human resources domain) including operations for totaling and cutting salaries (symbolized by the model element *Operations.cs*) as well as XML-related functionality for de-/serialization (see *Serialization.cs*). There are model elements for XML types according to the XSD language for XML schemas (see file *Company.xsd*) and *C#* classes (see file *Company.cs*) with fragments (see *Company*, *Department*, and *Employee*). There are correspondence relationships between the XML and object types to express that instances of these types can be (roughly) converted into each other (modulo the X/O impedance mismatch [18]). Class generation is automated with a batch file (see *CompanyXSD2CS.bat*), which essentially invokes the .NET tool `xsd.exe` (see *dependsOn*). Ultimately, the operation for cutting companies is invoked by

[5] MegaL is currently a combination of an ontology and a set of concrete syntaxes; there exist these flavors of the language: MegaL/yEd—a visual notation, MegaL/TXT—a textual notation and MegaL/RDF—an RDF version of MegaL. The correspondance between these notations is rather straightforward and it will be introduced by means of illustration in the course of the paper.

[6] http://msdn.microsoft.com/en-us/library/x6c1kb0s(v=vs.71).aspx

The upper frame uses the MegaL/yEd visual notation for megamodeling.
The lower frame shows *linked artifacts* of the product explained later in the paper.



```
<?xml version="1.0" encoding="UTF-8" star
<xs:schema xmlns="http://www.softlang.org

  <xs:element name="Company">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Name" type="xs:
        <xs:element name="TopLevelDepartn
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="Department">
    <xs:sequence>
      <xs:element name="Name" type="xs:st
      <xs:element name="Manager" type="Em
      <xs:element name="SubDepartment" ty
      <xs:element name="Employee" type="E
    </xs:sequence>
  </xs:complexType>
```

```
[System.Diagnostics.DebuggerStepThroughAt
[System.ComponentModel.DesignerCategoryAt
[System.Xml.Serialization.XmlTypeAttribut
[System.Xml.Serialization.XmlRootAttribut
public partial class Company {

    private string nameField;

    private Department[] topLevelDepartme

    /// <remarks/>
    public string Name {
        get {
            return this.nameField;
        }
        set {
            this.nameField = value;
        }
    }

    /// <remarks/>
```

Fig. 1. The linguistic architecture of a software product when displayed with the MegaL/Explorer tool.

demo functionality (see *Demo.cs*) and applied to a specific company—the *Acme Corporation*.[7]

The shown linguistic architecture describes artifacts as they arise during development time and runtime together with the relationships regarding dataflow, language membership, schema/type conformance, and correspondence. Charac-

[7] http://en.wikipedia.org/wiki/Acme_Corporation

teristics of the .NET technology for Object/XML mapping are clearly identifiable. Consider, for example, the fact that the class generator is not described as generating 'arbitrary' $C\#$. Instead, the subset *CSharpFromXsd* is introduced for referring to regular $C\#$ as produced by the generator. The identification of such 'hidden' languages is fundamental to the understanding of software technologies.

3 Entity and relationship types for megamodels

The proposed form of megamodeling essentially involves the identification and classification of entities and relationships that make up the linguistic architecture of software products or the underlying software technologies. In this section, we gather a set of entity and relationship *types* that may be used in megamodels.

3.1 Background

There exist megamodel-like models in different areas of computer science. Linguistic relations have been of interest since the early days of computing as *tombstone diagrams* testify. In Figure 2, on the left, we show a tombstone diagram, as it is used in compiler construction to describe the bootstrapping process for a C compiler, also written in the programming language C and compiling to M (the machine language) such that initially another C compiler is needed—this time written (or executable) in M. Hence, *languages* and *compilers* serve as entities while relationships are concerned with *dataflow* or *function application* and *membership*.

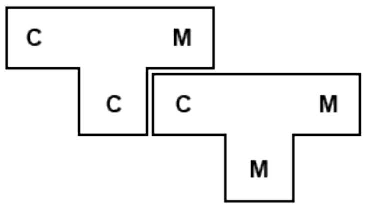
On the right, we show a much more recent diagram, as it appears in the documentation of the *ATL* transformation language; the diagram shows the mechanics of a model transformation in terms of entities for the involved *models* and *metamodels* as well as relationships for *conformance* and *dataflow*.

Further inspiration, specifically regarding linguistically relevant relationships, can be drawn from fundamental research on modeling and model management. The 'conformsTo' relationship is established in modeling for relating models and metamodels [9, 16]. We also rely on yet other basic modeling relationships (as in UML)—in particular 'partOf' and 'dependsOn'. The 'elementOf' and 'subsetOf' relationships are hardly used directly in regular modeling, but it appears in fundamental discussions, when *the usage of languages* is taken into account as opposed to sole restriction to metamodel-based conformance [9, 16]. The 'modelOf' or 'representationOf' relationship [16, 23, 24] is important for capturing the roles of descriptions, definitions, specifications, programs, or more generally models in megamodels. Ideally bidirectional intermodel mappings [5, 6], with interpretations at both the schema (metamodel) and the instance (model) level, give rise to the 'correspondsTo' relationship in our terminology.

Based on this background, the MegaL ontology defines a set of entity and relationship types as discussed below.

[8] Source: http://en.wikipedia.org/wiki/Tombstone_diagram
[9] Source: http://wiki.eclipse.org/ATL/Concepts#Model_Transformation

A tombstone diagram for bootstrapping a C compiler[8]



Mechanics of an ATL-based model transformation[9]

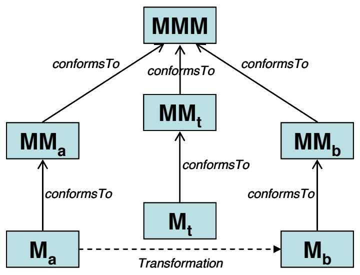


Fig. 2. Megamodels in different areas of computer science.

3.2 Entity types of MegaL

We distinguish three kind of entities: *abstract* entities—they appear at the mathematical level of thinking; *conceptual* entities—they are cognitive elements such as languages or technologies; *digital* entities—they correspond to artifacts that reside in and are processed by computers.

In this paper, we use these types of *abstract entities*: Entity, Set, Pair, Relation, Function, FunctionApplication (i.e., pairs pertaining to a function). For instance, functions are needed to model the meaning of tools or programs. Further, we use these types of *conceptual entities*: Language and Technology. Languages can be viewed (in a simplified manner) as sets. Technologies can be viewed as compound entities with components for tools, languages, and others. Finally, we use these types of *digital entities*: Artifact (the base type for the following types), File, Fragment (of a file), Program, Library, ObjectGraph.

The aforementioned entity types are just sufficient for the examples in this paper. The megamodel ontology can be extended to cover different domains, technological spaces, or engineering activities [8]. For instance, a megamodel in the context of model-driven engineering may benefit in clarity from additional digital entity types for models, metamodels, and model transformations.

3.3 Relationship types of MegaL

Based on the fundamental relationships and the types of entities, as identified above, the following relationship types can be derived. Again, the list is trimmed down for the scope of this paper. We apply a UML-like convention to use ':Type' for a concrete (anonymous) entity of the given type.

- ◇ :Language **subsetOf** :Language
- ◇ :Artifact **elementOf** :Language
- ◇ :Language **domainOf** :Function
- ◇ :Function **hasRange** :Language
- ◇ :FunctionApplication **elementOf** :Function

◇ :Artifact **inputOf** :FunctionApplication
◇ :FunctionApplication **hasOutput** :Artifact
◇ :Artifact **conformsTo** :Artifact
◇ :Artifact **partOf** :Artifact
◇ :Artifact **correspondsTo** :Artifact
◇ :Artifact **dependsOn** :Artifact
◇ :Artifact **dependsOn** :Language
◇ :Artifact **realizationOf** :Function
◇ :Artifact **definitionOf** :Language
◇ :Program **partOf** :Technology
◇ :Library **partOf** :Technology

Megamodels initially just *declare* entities and relationships. Eventually, megamodels may be *linked* so that both entities and relationships are meaningfully demonstrated by actual artifacts of specific software products. This will be discussed in §6.

4 An initial megamodel for O/X mapping

Megamodeling is demonstrated in this section for O/X mapping. In (schema-first) O/X mapping [19, 25], one is concerned with marrying object-oriented programming with XML-based data representation such that an object model for data representation is generated from an XML schema and library functionality is responsible for mediating between XML documents ('files') and objects back and forth. The population of objects from XML data is also called deserialization whereas the other direction is referred to as serialization. The notion of O/X mapping is also known as XML data binding. In the context of the .NET platform, the term XML serialization is used as well.

4.1 Stepwise development of the megamodel

Let us develop an initial megamodel for O/X mapping, step by step. We use MegaL/TXT—this simple textual notation can express the same concepts as the visual notation MegaL/yEd that we used earlier. The textual notation comes with straightforward syntactic shorthands for recurring patterns [8] such as '$\rightarrow$' and '$\mapsto$' (instead of combinations of 'domainOf', 'hasRange', 'inputOf', 'hasOutput').

We begin with the *languages* involved in O/X mapping:

Languages *XSD, CSharp, XML, ClrObjectGraphs .*

The *C#* (or *CSharp*) language is mentioned because it is assumed here that schema-derived object models are represented in *C#*. We could make the object-oriented programming language a parameter of the megamodel, but we commit to *C#* here for concreteness' sake. *XSD* is the language of XML schemas. *XML* is the language of XML trees (or XML documents), i.e., the primary ('on file') representation format for data. Finally, *ClrObjectGraphs* is the language of object graphs. Again, we could make the in-memory representation of objects a parameter of the megamodel, but we commit to .NET's CLR representation here for concreteness' sake.

In fact, another language should be identified:

Language *CSharpFromXsd* ***subsetOf*** *CSharp* .

That is, *CSharpFromXsd* proxies for the C# subset that is used by the class generator of the O/X mapping technology. In conservative discussions of O/X mapping, this language is never articulated. However, awareness of this language and its characteristics helps understanding O/X mapping.

The characteristics of schema-derived object models vary indeed for each O/X mapping technology. In the case of .NET's O/X mapping technology, we can state the following characteristics for all $x \in \mathit{CSharpFromXsd}$: (i) x declares classes only—as opposed to interfaces, enumerations, etc. (ii) The classes of x declare fields and properties as members, but no methods. (iii) x use attributes controlling XML serialization.

Let us now consider the major artifacts involved in O/X mapping. There are two type-level artifacts involved in such O/X mapping: an XML schema and an object model. There are also two instance-level artifacts involved: an actual XML document and an actual object graph:

File *xmlTypes* ***elementOf*** *XSD* .
File *ooTypes* ***elementOf*** *CSharpFromXsd* .
File *xmlDoc* ***elementOf*** *XML* .
ObjectGraph *clrObj* ***elementOf*** *ClrObjectGraphs* .

We also need to impose 'conformsTo' relationships as constraints on the instance-level artifacts: an arbitrary XML document would not be suitable; it must conform to the XML schema at hand; likewise for the object graph. Thus:

xmlDoc ***conformsTo*** *xmlTypes* .
clrObj ***conformsTo*** *ooTypes* .

Ultimately, we expect an O/X mapping technology to provide functionality for class generation and for deserialization (as well as serialization, which we skip here though). To this end, we introduce the following conceptual entities, in fact, functions, and we apply them in the expected manner to relate the artifacts at the type and instance levels. Thus:

Function *classgen* : *XSD* $\rightarrow$ *CSharpFromXsd* .
Function *deserialize* : *XML* $\rightarrow$ *ClrObjectGraphs* .
classgen(xmlTypes) $\mapsto$ *ooTypes* .
deserialize(xmlDoc) $\mapsto$ *clrObj* .

4.2 Summary of the megamodel

Figure 3 summarizes the megamodel in the form of a diagram drawn with the MegaL/yEd editor.[10] The visual and the textual notation convey the same in-

[10] Our implementation uses yEd for megamodel editing http://www.yworks.com/en/products_yed_about.html with GraphML http://graphml.graphdrawing.org/ for the representation.

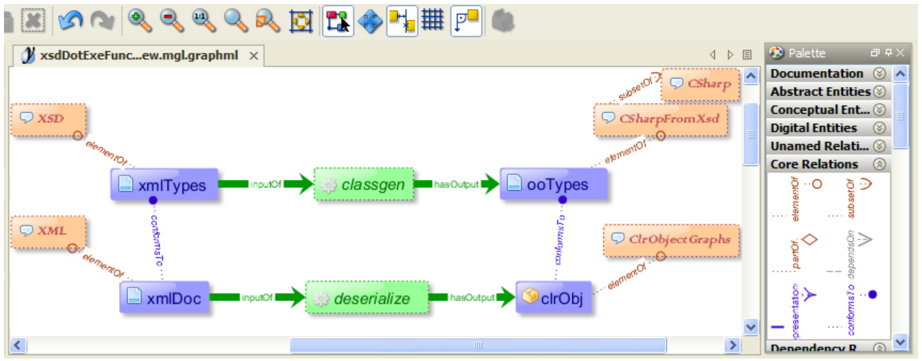


Fig. 3. An initial megamodel for O/X mapping drawn with the MegaL/yEd editor.

formation. Note that icons and colors are bound to entity types in the diagram. Some megamodel elements can be mapped to different visual elements. For instance, ‘partOf’ relationships are represented by node embedding in the upper frame of Figure 1, but a regular ‘partOf’ edge could also be used.

4.3 Discussion

The initial megamodel of this section was deliberately kept simple. This intermediate state also allows us to reflect on methodological questions of megamodeling:

- Do we model all important aspects of O/X mapping overall?
- What specifics of .NET’s O/X mapping technology should be modeled?

Without focus on O/X mapping, these questions take the following form:

- Do we model all general aspects of the kind of technology at hand?
- What specifics of a concrete technology should be modeled?

It is relatively easy to observe that the megamodel could be enhanced to incorporate additional aspects of O/X mapping, overall. For instance, we did not yet model the fact that O/X mapping is carried out ‘for a purpose’: some OO program is meant to use the generated object model to implement data-processing functionality. As to the question of technology-specific aspects, we did not yet model the components of .NET’s technology for O/X mapping. These and additional aspects are addressed in the following section.

5 A megamodel for O/X mapping with .NET

We advance the megamodel of the previous section to cover generally more aspects of O/X mapping and to also apply more directly to the situation for the .NET platform.

5.1 The use of schema-derived object models

The value proposition of O/X mapping depends on the fact that it enables essentially OO programming on XML data. We capture this aspect in the megamodel by introducing a problem-specific program that is said to depend on the schema-derived object model. This is another placeholder for an entity that does not belong to the technology itself, but instead to the software product that uses the technology. Thus:

File *problemProgram* ***elementOf*** *CSharp .*
problemProgram ***dependsOn*** *ooTypes .*

5.2 Technology components for .NET

The technology consists of a code-generation tool, `xsd.exe`, a library, hosted by the namespace *System.Xml.Serialization*, and custom attributes (annotations) for metadata.[11] We declare corresponding entities:

Program *xsdDotExe . −− the "xsd.exe" tool*
Library *XmlSerializer . −− namespace "System.Xml.Serialization"*
Language *XsdMetadata .*

We can model now the fact that the `xsd.exe` tool realizes the class generation functionality for O/X mapping. In fact, the tool also realizes additional functionality, e.g., related to O/R mapping. To this end, the tool can be used in different modes controlled through the command line or an API, but we do not model such variability here. Thus:

xsdDotExe ***realizationOf*** *classgen .*

Previously, we simply assumed a function, *deserialize*, for deserializing XML into objects, without though clarifying the origin of the function. It is the problem-specific program that essentially performs de-serialization. In fact, we assume that some part of the program realizes serialization by making appropriate use of .NET's library for XML serialization. Thus:

Fragment *deserialization* ***partOf*** *problemProgram .*
deserialization ***dependsOn*** *XmlSerializer .*
deserialization ***realizationOf*** *deserialize .*

We can also clarify the role of metadata in O/X mapping. We assume that, subject to an appropriate interpretation of 'partOf' for languages, the *C#* language indeed comprises a part for metadata such that metadata for O/X mapping is a subset of general metadata.

Language *CSharpMetadata .*
CSharpMetadata ***partOf*** *CSharp .*
XsdMetadata ***subsetOf*** *CSharpMetadata .*

[11] http://msdn.microsoft.com/en-us/library/ms950721.aspx

Also, we can capture the characteristics of schema-derived classes to depend on metadata for O/X mapping. We do not formalize other characteristics of *CSharpFromXsd*. Thus:

ooTypes ***dependsOn*** *XsdMetadata* .

5.3 Additional linguistic details

Let us call *problemLanguage*a problem-specific language underlying the involved type-level artifacts. We think of this language as being abstract, rather than concretely represented by XML trees or object graphs. This language can be viewed as a proxy for the domain that is covered with a Object/XML mapping effort.

Language *problemLanguage* .
xmlTypes ***definitionOf*** *problemLanguage* .
ooTypes ***definitionOf*** *problemLanguage* .

It remains to establish a correspondence relationship between XML and object types as well as the involved instances:

xmlTypes ***correspondsTo*** *ooTypes* .
xmlDoc ***correspondsTo*** *clrObj* .

At the instance level, the object graph, which is obtained by de-serialization, is expected to be a *representation of* the original XML document and *vice versa* such that the original document could be re-obtained by serialization from which we abstract here for simplicity.

At the type level, correspondence means that (ideally) XML schema and object model are related by bidirectional intermodel mappings (say, 'structure-preserving' bijections) modulo difficulties due to the O/X impedance mismatch [18]. The couple of de-serialization and serialization functionalities should be considered the concrete interpretation of these mappings at the instance level, but this view is not developed in detail here. More intuitively, we could say that there is 1:1 mapping of types driven by name equality or similarity, and for each couple of associated types there is also a correspondence at the 'member' level.

5.4 Discussion

We conclude with a discussion of potential directions for enhancing the megamodel. We have focused here on de-serialization, but serialization could also be of interest, if XML transformation or generation is to be modeled. Further, we have not modeled any variability or configurability admitted the mapping technology, as needed for advanced usage scenarios of the technology.

We claim originality for analyzing O/X mapping by megamodeling. For comparison, the arguably most comprehensive catalog of O/X mapping technologies [25] uses an informal metamodel to compare technologies (tools) on the grounds of capabilities and limitations—linguistically relevant entities and relationships are not considered.

6 Linked megamodels

A difficulty with metamodeling and even more with megamodeling approaches resides in the high level of abstraction they involve. This difficulty is even exacerbated by megamodels that deal with technologies, as in the previous two sections, because of the gap between the abstract notation and the very concrete artifacts a software engineer deals with, e.g., some files or objects. As a result it may be hard to convince anyone that any given statement in the megamodel holds.

Linked megamodels close the gap between abstraction and concreteness by linking each entity in the megamodel to a web resource. Thus, an entity is no longer represented merely as an identifier, leaving all room for misunderstanding and misinterpretation; instead, the identifier is linked to a unique resource that can be browsed and examined at will. Relationships can also be linked. As a result, it becomes much easier to understand and to validate megamodels.

6.1 Binding placeholder entities

Note that in the megamodel of the previous two sections, artifact placeholders were used for some entities, e.g., *xmlDoc* and *clrObj*. When the goal is to validate or illustrate the megamodel, then it is useful to 'bind' placeholders to actual artifacts. This has been done in Figure 1 with the concrete artifacts being part of a particular software product. That is, X/O mapping is illustrated thanks to the *xsdClasses* implementation of the *101companies* project. For instance, the placeholder *xmlDoc* is bound to *Company.xsd*—an XML schema file of the *xsdClasses* implementation.

6.2 Exploring linked megamodels

From the end-user perspective, linked megamodels are seen as hypertext documents that can be explored. Figure 1 shows a screenshot of the MegaL/Explorer tool. The upper frame corresponds to a clickable image that is produced with MegaL/Editor. Within the context of the explorer, a click on an entity displays the corresponding resource in the lower frame. For instance, clicking on the *CSharp* node leads to a wiki page for C# according to the *101companies* project; clicking on `xsd.exe` node also leads to a page for the tool; clicking on a file, e.g., *Company.xsd*, displays the content of the file extracted from the *101companies* repository. Relationships (i.e., graph edges) are also clickable. In Figure 1, the user has selected the (circled) correspondence link between the *Company* fragments respectively in *Company.xsd* and *Company.cs*. As a result, the source fragments are shown side by side in the lower frame—clearly showing what the `xsd.exe` tool actually generates for a given example.

6.3 MegaL/RDF, linked megamodels and *Linked Data*

Technically, linked megamodels are represented in RDF by following *Linked Data*[12] principles. Figure 4 and Figure 5 show fragments of two megamodel

[12] http://linkeddata.org/

```
_:xmlTypes rdf:type mgl:File .
_:xmlTypes rdfs:label "xmlTypes" .
_:xmlTypes mgl:elementOf lang:XSD .
_:xmlTypes mgl:inputOf _:classgen .

_:xmlDoc rdf:type mgl:File .
_:xmlDoc rdfs:label "xmlDoc" .
_:xmlDoc mgl:elementOf lang:XML .
_:xmlDoc mgl:conformsTo _:xmlTypes .
_:xmlDoc mgl:inputOf _:classgen .

_:classgen_app_1 rdf:type mgl:FunctionApplication .
_:classgen_app_1 rdfs:label "classgen" .
_:classgen_app_1 rdf:elementOf _:classgen .
_:classgen_app_1 rdf:hasOutput _:ooTypes .

... etc. ...
```

Entities are associated with a prefix, e.g., `rdf`, corresponding to a unique URI (not shown here). This means that each entity is now associated with a URL where the corresponding resource can be found. Only 'blank nodes', i.e., those with the _ prefix, are local identifiers. The `rdf` and `rdfs` prefixes refers to RDF and RDFS definitions respectively. The prefix `mgl` refers to the MegaL ontology which contains definitions for both entity types (represented as OWL classes) and relationships types (represented as OWL properties).

Fig. 4. Figure 3 expressed in MegaL/RDF.

```
_:CompanyDotXSD rdf:type mgl:File .
_:CompanyDotXSD rdfs:label "Company.xsd" .
_:CompanyDotXSD mgl:elementOf lang:XSD .
_:CompanyDotXSD mgl:inputOf _:CompanyXSD2CSDotBat .
_:CompanyElement mgl:partOf _:CompanyDotXSD .
_:CompanyElement rdf:type mgl:FileFragment .
_:CompanyElement rdfs:label "Company" .
... other fragments omitted ...

_:CompanyDotXSD mgl:partOf impl:xsdClasses .
_:CompanyDotXSD mgl:filename "./Company.xsd" .
_:CompanyElement mgl:xpathLocation
                   "//*[@name=\"Company\"]" .
... etc
```

The first block of triples shows some properties of the file `Company.xsd` including its decomposition into fragments.

The second block models links to online software artifacts. For instance the `impl` prefix refers to *101companies* implementations, `./Company.xsd` refers to a file name, and the property `mgl:xpathLocation` refers to a fragment of the schema file.

Fig. 5. RDF-based links for the megamodel of Figure 1.

expressed in MegaL/RDF as sets of triples while using RDF/turtle syntax. The first figure is concerned with the general megamodel for O/X mapping. It contains therefore placeholders with generic names, e.g. *xmlDoc* and *xmlTypes*). By contrast, the second figure is concerned with the bound megamodel for the *101companies* implementation *xsdClasses*. It contains product-specific names, e.g. , *CompanyDotXSD*, but also, and this is a very important aspect, links to concrete software artifacts, which should be considered as *resources* according to RDF principles.

Since all information in the *101companies* project is represented as RDF triples, links between *101companies* resources and external ones such as Wikipedia pages, i.e., dbpedia[13] resources in terms of RDF, this approach therefore enables the integration of megamodels and various other resources in the *Linked Data* global data space.

[13] http://dbpedia.org

7 Related work

Megamodeling Megamodeling is somewhat established in the communities of modeling and model-driven engineering. Existing forms of megamodels do not cover the range of linguistic relationships of MegaL (such as 'elementOf', 'subsetOf', and 'correspondsTo'); they have not been used in a manner to understand software technologies across technological spaces. We look at representative examples. In [27], megamodeling is applied to the human-computer interaction domain. In [10], a UML/OCL-based megamodel of MDA/MDE is provided, thereby supporting reasoning about MDA/MDE. In [30], megamodeling is used for organizing and utilizing runtime models and relations in a model-driven manner while also supporting a high level of automation. In [15], megamodeling is used to coordinate "heterogeneous" models in the sense of conforming to a multiplicity of metamodels expressed in different DSLs. In [13], megamodeling is applied to model transformation with the objective of supporting the evolution of software architectures. In [14], some forms of megamodels and associated applications are surveyed.

Some model transformation approaches involve explicitly chains or compositions of transformations, perhaps even involving different model transformation languages and dealing with different 'modeling spaces'. Such compositions can be viewed as a form of executable megamodels. In [20], the authors motivate the need for a precise semantics for model-to-model transformations, thereby enabling verification of correctness for compositions, thereby, in turn, encouraging reusability.

Foundations of modeling Our work is substantially inspired by recent efforts on the foundation of modeling from which we derive basic idioms of megamodeling. We rely on established relationships such as 'conforms to' and 'element of' [9,16]. Further, there is the multi-faceted 'represents/models' relation [23,24]. We derive the correspondence relation from the field of model management. In [5,6], a categorical approach to intermodel mappings including heterogeneous (meta)model correspondences is developed.

Viewpoints We consider megamodels as supporting another 'point of view' in the tradition of viewpoints in software development [12]. Viewpoints are used in practice, specifically for enterprise architecture [28], on the basis of the reference model RM-ODP[14] and the IEEE-1471 standard[15]. Each viewpoint is typically associated with one or more designated modeling languages [4] subject to different metaware [8] (i.e., metamodels and model-driven software technology). In this paper, we enable the linguistic point of view.

[14] http://www.rm-odp.net/

[15] http://standards.ieee.org/findstds/standard/1471-2000.html

8 Conclusion

We have developed a form of modeling that targets the linguistic architecture of software technologies and software products. Megamodels serve as cognitive models for the benefit of software engineers, software linguists, and others.

We expect to advance the *101companies* project to provide megamodels systematically for a substantial number of software technologies and *101companies* contributions. We plan to use such megamodels in teaching software technologies to undergraduates. Without megamodels, it is very difficult to convey sufficiently abstract knowledge about, for example, technologies for Object/Relational/XML mapping in university courses.

Future research should advance the megamodeling approach in several dimensions. Megamodels should be extended to incorporate declarative descriptions of relationships for conformance, correspondence, membership, and others so that the cognitive value of such extended megamodels is improved. For instance, newly identified languages, such as the *C#* subset used by the generator in our example, could be properly described in this manner. Also, our understanding of intermodel mappings, such as the mapping between XML and object types, could be properly explained in this manner. Such descriptions could leverage language support for code queries.

The notion of linked megamodels will be advanced so that some links can be recovered semi-automatically from products that adhere to some tagging and naming rules. Also, static and dynamic program analysis will be leveraged so that the applicability of a generic megamodel to a specific product can be verified and eventually inferred.

Acknowledgement The authors are grateful to a number of important discussions and exchanges with D. Gasevic and Z. Diskin that helped developing the present notion of megamodeling. An emerging notion was presented by the authors on different occasions at SLE/GPC 2011 (tutorial by J.M. Favre, D. Gasevic, and R. Lämmel), GTTSE 2011 (short participant presentation by A. Varanovich without publication), AOSD 2011 (tutorial by R. Lämmel), KPS 2011 (informal workshop presentation by R. Lämmel). The authors are grateful to several students—Martin Leinberger, Marius Rackwitz, and Thomas Schmorleiz—for their participation in this research.

References

1. Bézivin, J., Jouault, F., Valduriez, P.: On the need for Megamodels. In: Proceedings of the OOPSLA/GPCE: Best Practices for Model-Driven Software Development workshop (2004)
2. Bézivin, J., Jouault, F., Rosenthal, P., Valduriez, P.: Modeling in the Large and Modeling in the Small. In: Model Driven Architecture, European MDA Workshops: Foundations and Applications, MDAFA 2003 and MDAFA 2004, Revised Selected Papers. LNCS, vol. 3599, pp. 33–46. Springer (2005)
3. Bräuer, M., Lochmann, H.: An Ontology for Software Models and Its Practical Implications for Semantic Web Reasoning. In: The Semantic Web: Research and Applications, 5th European Semantic Web Conference, ESWC 2008. Proceedings. LNCS, vol. 5021, pp. 34–48. Springer (2008)

4. Dijkman, R.M., Quartel, D.A.C., Pires, L.F., van Sinderen, M.: An Approach to Relate Viewpoints and Modeling Languages. In: 7th International Enterprise Distributed Object Computing Conference (EDOC 2003), Proceedings. pp. 14–27. IEEE (2003)
5. Diskin, Z., Maibaum, T., Czarnecki, K.: Intermodeling, Queries, and Kleisli Categories. In: Fundamental Approaches to Software Engineering - 15th International Conference, FASE 2012. Proceedings. LNCS, vol. 7212, pp. 163–177. Springer (2012)
6. Diskin, Z., Xiong, Y., Czarnecki, K.: Specifying Overlaps of Heterogeneous Models for Global Consistency Checking. In: Models in Software Engineering - Workshops and Symposia at MODELS 2010, Reports and Revised Selected Papers. LNCS, vol. 6627. Springer (2011)
7. Djuric, D., Gasevic, D., Devedzic, V.: The Tao of Modeling Spaces. Journal of Object Technology 5(8) (2006)
8. Favre, J.M.: CacOphoNy: Metamodel-Driven Architecture Recovery. In: 11th Working Conference on Reverse Engineering (WCRE 2004), Proceedings. pp. 204–213. IEEE (2004)
9. Favre, J.M.: Foundations of meta-pyramids: Languages vs. metamodels – Episode II: Story of thotus the baboon. In: Language Engineering for Model-Driven Software Development. No. 04101 in Dagstuhl Seminar Proceedings (2005)
10. Favre, J.M.: Foundations of Model (Driven) (Reverse) Engineering: Models – Episode I: Stories of The Fidus Papyrus and of The Solarus. In: Language Engineering for Model-Driven Software Development. No. 04101 in Dagstuhl Seminar Proceedings (2005)
11. Favre, J.M., Lämmel, R., Schmorleiz, T., Varanovich, A.: *101companies*: a community project on software technologies and software languages. In: Proceedings of TOOLS 2012. LNCS, Springer (2012), 16 pages. To appear.
12. Finkelstein, A., Kramer, J., Nuseibeh, B., Finkelstein, L., Goedicke, M.: Viewpoints: A framework for integrating multiple perspectives in system development. International Journal of Software Engineering and Knowledge Engineering 2(1), 31–57 (1992)
13. Graaf, B.: Model-Driven Evolution of Software Architectures. Dissertation, Delft University of Technology (2007)
14. Hebig, R., Seibel, A., Giese, H.: On the Unification of Megamodels. In: Proceedings of the 4th International Workshop on Multi Paradigm Modeling (MPM'10) at the 13th IEEE/ACM International Conference on Model Driven Engineering Languages and Systems (MoDELS 2010) (2010)
15. Jouault, F., Vanhooff, B., Bruneliere, H., Doux, G., Berbers, Y., Bézivin, J.: Inter-dsl coordination support by combining megamodeling and model weaving. In: SAC. pp. 2011–2018 (2010)
16. Kühne, T.: Matters of (Meta-) Modeling. Software and Systems Modeling 5, 369–385 (2006)
17. Kurtev, I., Bézivin, J., Aksit, M.: Technological spaces: An initial appraisal. In: CoopIS, DOA'2002 Federated Conferences, Industrial track (2002)
18. Lämmel, R., Meijer, E.: Revealing the X/O impedance mismatch (Changing lead into gold). In: Spring School on Datatype-Generic Programming, Lecture Notes. LNCS, vol. 4719, pp. 285–367. Springer (2007)
19. Lämmel, R., Meijer, E.: Mappings Make Data Processing Go 'Round. In: Generative and Transformational Techniques in Software Engineering, International Summer School, GTTSE 2005. Revised Papers. LNCS, vol. 4143, pp. 169–218. Springer (2006)

20. Lano, K., Rahimi, S.K.: Model-driven development of model transformations. In: Theory and Practice of Model Transformations, Fourth International Conference, ICMT 2011, Zurich, Switzerland, June 27-28, 2011. Proceedings. pp. 47–61. Lecture Notes in Computer Science, Springer (2011)
21. Mah/'e, V., Perez, S.M., Brunelière, H., Doux, G., Cabot, J.: PORTOLAN: a Model-Driven Cartography Framework. Tech. rep., INRIA (2011), # 7542
22. Melnik, S., Adya, A., Bernstein, P.: Compiling mappings to bridge applications and databases. In: SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data. pp. 461–472. ACM (2007)
23. Muller, P.A., Fondement, F., Baudry, B.: Modeling Modeling. In: Model Driven Engineering Languages and Systems, 12th International Conference, MODELS 2009, Proceedings. LNCS, vol. 5795, pp. 2–16. Springer (2009)
24. Muller, P.A., Fondement, F., Baudry, B., Combemale, B.: Modeling modeling modeling. Software and Systems Modeling pp. 1–13 (2011)
25. Ronald Bourret: Xml data binding resources (2001–2012), http://www.rpbourret.com/xml/XMLDataBinding.htm
26. Salay, R., Mylopoulos, J., Easterbrook, S.M.: Using Macromodels to Manage Collections of Related Models. In: Advanced Information Systems Engineering, 21st International Conference, CAiSE 2009. Proceedings. LNCS, vol. 5565, pp. 141–155. Springer (2009)
27. Sottet, J.S., Calvary, G., Favre, J.M., Coutaz, J.: Megamodeling and Metamodel-Driven Engineering for Plastic User Interfaces: MEGA-UI. In: Human-Centered Software Engineering. pp. 173–200. Springer Human-Computer Interaction Series (2009)
28. Steen, M.W.A., Akehurst, D.H., ter Doest, H.W.L., Lankhorst, M.M.: Supporting Viewpoint-Oriented Enterprise Architecture. In: 8th International Enterprise Distributed Object Computing Conference (EDOC 2004), Proceedings. pp. 201–211. IEEE (2004)
29. Thomas, D.: The Impedance Imperative: Tuples + Objects + Infosets = Too Much Stuff! Journal of Object Technology 2(5), 7–12 (Sep–Oct 2003)
30. Vogel, T., Seibel, A., Giese, H.: The Role of Models and Megamodels at Runtime. In: Models in Software Engineering - Workshops and Symposia at MODELS 2010, Reports and Revised Selected Papers. LNCS, vol. 6627, pp. 224–238. Springer (2011)