Axioms of Linguistic Architecture

Marcel Heinz, Ralf Lämmel and Andrei Varanovich
Software Languages Team, University of Koblenz-Landau, Universitätsstraße 1, 56072 Koblenz, Germany
http://www.softlang.org/

Keywords: Software Technology, Software Language, Axiomatization, Ontology, Linguistic Architecture, Megamodeling.

Abstract: In documenting software technologies (e.g., web application or modeling or object/relational mapping frameworks) and specifically when discussing technology usage scenarios, one aims at identifying and classifying the involved entities (e.g., languages and artifacts); one also aims at relating the entities (e.g., through conformance or I/O behavior of program execution). In this paper, we present a logic-based axiomatization (an emerging ontology) for the underlying types of entities and relationships, thereby formalizing recurring documentation idioms such as '*a software system (e.g., a Java application) to use a technology (e.g., a test library)*' or '*a technology (e.g., a web application framework) to facilitate a certain concept (e.g., the MVC pattern)*'. The axiomatization is illustrated by examples applying to the Eclipse Modeling Framework. The inclusion of types of entities and relationships is driven and thus validated by a literature survey on megamodeling.

1 INTRODUCTION

Research Context: *Linguistic Architecture*. This work should be regarded as feeding into the 'Software Engineering Body of Knowledge'[1] in the context of modeling *software language and technology usage* as a form of knowledge aggregation in the software engineering field (Ruiz and Hilera, 2006). In previous work, modeling such usage is seen as a form of megamodeling (Bézivin et al., 2004; Diskin et al., 2013), subject to models of *linguistic architecture* (Favre et al., 2012a)—this term emphasizes languages ('linguae') because of the central role of language-typed artifacts in the (mega-) models.

Research Objective: *Technology Documentation*. The broader objective of our research on linguistic architecture is to assist comprehension (i.e., understanding) of software technologies (e.g., model transformations, object/relational mappers, or web-application frameworks). Models of linguistic architecture correspond to disciplined documentation of technologies built from conceptual facts conforming to an appropriate schema. We assume that such a discipline improves the quality of documentation and makes it more useful, e.g., when software engineers study a new technology to be used in a project. The megamodeling-based approach should be suitable to describe or prescribe usage of languages and technologies in a more precise manner than the common informal and ad-hoc approach to documentation.

Research Contribution: *An Axiomatization*. In this paper, we axiomatize central model elements of linguistic architecture. In essence, we provide a reference semantics for conceptual facts to be used in models. In different terms, we provide a well-defined vocabulary for documentation as opposed to relying on an intuitive understanding of the vocabulary. That is, we axiomatize key relationship types, thereby formalizing recurring documentation idioms such as '*a software system (e.g., a Java application) to use a technology (e.g., a test library)*' or '*a technology (e.g., a web application framework) to facilitate a certain concept (e.g., the MVC pattern)*'.

The axiomatization is illustrated systematically by examples applying to EMF[2]. The axiomatization combined with the illustrations feed into an emerging ontology for *Software Languages and Software Technologies* to which we refer as SoLaSoTe[3] in the rest of the paper. The inclusion of types of entities and relationships is driven and thus validated by a literature survey on megamodeling. That is, the survey justifies the selection of axiomatized model elements and shows the broader impact of this work. Our work is

[1] https://www.computer.org/web/swebok

[2] https://www.eclipse.org/modeling/emf/

[3] http://www.softlang.org/solasote

Heinz M., LÃd'mmel R. and Varanovich A.
Axioms of Linguistic Architecture.
DOI: 10.5220/0006210204780486
In *Proceedings of the 5th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2017)*, pages 478-486
ISBN: 978-989-758-210-3
Copyright © 2017 by SCITEPRESS – Science and Technology Publications, Lda. All rights reserved

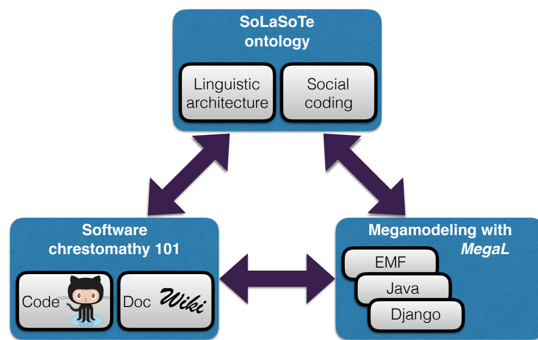

Figure 1: Ontology engineering process for SoLaSoTe.

also embedded into a process for ontology engineering aiming at better understanding usage of software languages and technologies.

Road-map of the Paper. Sec. 2 summarizes the underlying process for ontology engineering. Sec. 3 surveys research on megamodeling. Sec. 4 develops the axiomatization. Sec. 5 concludes the paper.

2 ONTOLOGY ENGINEERING

Our work on SoLaSoTe adopts the notion of ontology engineering (Corcho et al., 2006; Calero et al., 2006; Oberle et al., 2006; d'Aquin and Gangemi, 2011) through a process involving three pillars:

Chrestomathy. We have been contributing to the software chrestomathy '101companies' (or just '101') (Favre et al., 2012b)[4] which is a collection of small software systems that implement a common feature model while aiming at representing best practices and options of language usage, technology usage, and software design. The systems are documented on a semantic wiki; the documentation includes properties of language and technology usage.

MegaL. We have been designing megamodeling languages for linguistic architecture, most notably MegaL[5] (Favre et al., 2012a). The megamodels declare how 'digital' entities (such as files or objects) and 'conceptual' entities (such as languages or programming techniques) relate in the context of scenarios of technology and language usage. Such declarations can be verified (Lämmel and Varanovich, 2014).

SoLaSoTe. The ontology provides a framework for documentation of usage scenarios and actual systems. The ontology includes reusable facts or general axioms. There are two aspects: linguistic architecture—the focus of this paper—and social coding—an extension for developer roles and corresponding relationships not further discussed in this paper.

University courses, professional education, open-source development, summer schools, and scholarly work are used to advance 101, MegaL, or SoLaSoTe. These three pillars are mutually dependent; see Fig. 1. Progress at individual pillars and continuous reviewing help propagating knowledge about technology and language usage from pillar to pillar.

3 LITERATURE SURVEY

This section presents a survey with regard to the following research question: 'What kind of entity and relationship types exist in related work on megamodeling?'. Details and datasets are available from SoLaSoTe's website (see first page). The presented overview serves as a justification for the choice of the core vocabulary in the emerging ontology.

We searched for papers at 'ACM Digital Library' (ACM)[6], 'Springer Link' (Springer)[7] and 'IEEE Xplore Digital Library' (IEEE)[8] using the sites' search engines with the search string '*"mega model" OR "mega-model" OR "megamodel"*'. While ACM's and IEEE's default search settings only consider structured content (such as title, abstract and keywords), for Springer, we had to manually check search results for a match in the abstract, title or keywords while restricting the results to be in the software engineering category 'SWE'. We did not perform snowballing (Wohlin, 2014) to limit the amount of papers, as the analysis for paper inclusion is relatively laborious.

We screened the identified papers explicitly for relevance based on the following criteria. We *included* all papers that define types of megamodel elements in a dedicated section, a schematic notation, or a metamodel. We *excluded* explicit doubles and papers that only show language elements that are presented in a preceding paper.

We classified the entity and relationship types from the relevant papers. One paper (Favre et al., 2012a) was chosen to provide an initial set of classifiers for entity and relationship types. We incrementally updated the set by newly identified classifiers according to the typical process of a mapping study (Elberzhager et al., 2012). Table 1 and Table 2 presents

[4] http://101companies.org/
[5] http://www.softlang.org/megal
[6] http://dl.acm.org/
[7] http://link.springer.com/
[8] http://ieeexplore.ieee.org/Xplore/home.jsp

Table 1: Entity types in relevant papers.

Paper	Artifact	Function	Record	System	Technology	Language	Inf. resource	Fragment	Collection	Trace	Concept	Others
[1]	x	x	x				x					x
[2]	x	x	x		x					x		x
[3]	x			x	x						x	x
[4]					x	x	x				x	x
[5]	x						x	x		x		x
[6]	x		x									
[7]	x	x	x									
[8]	x									x		
[9]	x						x		x			
[10]	x	x	x		x	x		x	x			
[11]	x	x	x							x		
[12]				x								x
[13]	x	x								x		x
[14]	x	x										

Table 2: Relationship types in relevant papers.

Paper	Conformance	Definition	Correspondence	Implementation	Usage	Membership	Typing	Dependency	Abstract rel.	Others
[1]					x					x
[2]	x									
[3]	x	x	x	x	x			x		x
[4]				x	x		x	x		x
[5]				x			x			x
[6]						x	x		x	
[7]										x
[8]									x	
[9]		x							x	
[10]	x	x	x	x		x	x	x	x	
[11]	x	x					x		x	
[12]		x								x
[13]							x			
[14]	x								x	

the identified classifiers and their coverage by the papers. The classification's documentation including a glossary for the classifiers can be found online.

A few papers contain informal descriptions of additional entity types (see column 'Inf. resources'). We did not integrate these types, as they would be hard to validate. A few papers point out abstract relationships without concrete semantics (see column 'Abstract rel.') which we did not integrate either. 'Dependency' relationships are also not integrated since they can be expressed more explicitly in terms of 'Usage'. The column 'Others' states the appearance of entity and relationship types that are specific to a paper's megamodeling domain.

4 AXIOMATIZATION

When developers want to use technologies unknown to them, it is crucial for them to understand what a technology has to offer and how it is conceptually structured. In order to reach a high degree of understandability and precision for the vocabulary expressing such conceptual knowledge we present a formal axiomatization. We formulate the axioms here in predicate logic for ease of reading and brevity; see SoLaSoTe's website for mechanized versions.

For each group of axioms, we provide a natural language description and illustrative examples. The axiomatization starts with ontological classification in terms of subtyping as well as domain and range for relationships. Afterwards, integrity constraints as inspired by (Tran and Debruyne, 2012) are stated. We illustrate the usage of the predicates with scenarios for EMF[9]. Similar knowledge can also be gathered for other technological spaces (Kurtev et al., 2002), e.g., SQL-Ware or XML-Ware. An ontology based on the axiomatization may reuse defined vocabulary from an upper ontology such as DOLCE (Gangemi et al., 2002) that, e.g., already specifies part-hood.

4.1 Artifacts

Several disjoint subtypes of a root type *Entity* form the basis of the core vocabulary. The first such type is *Artifact* with digital entities as instances. We distinguish subtypes of *Artifact*: files and folders are represented as instances of the types *File* and *Folder*. Files and folders may not only appear in the local file system but on a website, subject to the subtype *WebResource*. Further, we introduce the subtype *Transient* for artifacts that only exist during program execution. Finally, we introduce the subtype *Fragment* for artifacts that only exist as parts of other artifacts. (A fragment cannot be a file or folder at the same time.)

Whether something is defined as an instance of *Artifact* or one of the subtypes depends on the chosen level of abstraction. A database can either be in a single file or scattered over a folder. Thus, one may choose to only define it as an *Artifact* without choosing a specific subtype. An artifact can have multiple types. We introduce a set of illustrative artifacts in Table 3 to which we will relate in the rest of the paper. In tables like this, we provide the exemplary entities or relationships in the left column and an informal description in the right column.

[9]http://eclipsesource.com/blogs/tutorials/emf-tutorial/

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

Table 3: Exemplary artifacts for EMF.

Artifacts	Comments
org.eclipse.emf.ecore	This folder implements the metametamodel
org.eclipse.emf.ecore.EObject	EObject is the root type of all modeled objects
org.eclipse.emf.ecore.EPackage	A type for packages
MyMeta.ecore	A metamodel written in Ecore
MyMeta.genmodel	A configuration file for code generation
MyMeta.genmodel.ref	A fragment referencing the metamodel from which code is generated
metaname	A fragment defining a name for the metamodel
MyModel	An transient model as a Java Object
MyModel.xmi	A model serialized in XMI

$$Artifact(e) \Rightarrow Entity(e).$$
$$File(a) \Rightarrow Artifact(a).$$
$$Folder(a) \Rightarrow Artifact(a).$$
$$WebResource(a) \Rightarrow Artifact(a).$$
$$Transient(a) \Rightarrow Artifact(a).$$
$$Fragment(a) \Rightarrow Artifact(a) \wedge \neg(File(a) \vee Folder(a)).$$

4.2 Systems and Technologies

We introduce two basic subtypes representing software: *System* and *Technology*. A system is a collection of artifacts supporting some use cases. Technologies, e.g., frameworks and libraries, provide functionality meant to be reused in different systems—possibly several times in each system.

$$System(e) \Rightarrow Entity(e).$$
$$Technology(e) \Rightarrow Entity(e).$$

A technology, system or an artifact can be composed of artifacts. Such a composition relationship is for example defined in DOLCE as 'partOf'. If the composite in a part-of-relationship is a fragment, all parts have to be fragments as well. Examples for systems, technologies, fragments, and composition relationships are provided in Table 4.

$$Fragment(f) \Rightarrow \exists a. Artifact(a) \wedge partOf(f,a).$$
$$partOf(p,w) \wedge Fragment(w) \Rightarrow Fragment(p).$$

4.3 Definition and Implementation

We introduce *Specification* as an artifact type that defines functional and non-functional requirements, which are implemented by a system or technology. A given specification can be implemented by several systems or technologies which thus may count as variants of each other.

Instances of *Function* represent functions that map input artifacts to output artifacts. A function is often defined in a specification and then implemented in a technology or a system. On a more detailed level, an artifact contains the code that actually implements the function.

In our sense, a *Language* instance is a set of syntactic entities which may be represented as artifacts. Such sets are defined by specifications or implemented by a technology. A specification may be a grammar or a web document, e.g., the UML superstructure[10]. A typical example for an implementing technology is a compiler. Further examples for such definition and implementation relationships can be found in Table 4.

$$Specification(a) \Rightarrow Artifact(a).$$
$$Function(e) \Rightarrow Entity(e).$$
$$Language(e) \Rightarrow Entity(e).$$
$$defines(a,e) \Rightarrow Artifact(a) \wedge Entity(e).$$
$$implements(x,y) \Rightarrow (Artifact(x) \vee System(x) \vee Technology(x)) \wedge Function(y) \vee Technology(x) \wedge Language(y).$$
$$Language(l) \Rightarrow (\exists s. Specification(s) \wedge defines(s,l)) \vee (\exists t. Technology(t) \wedge implements(t,l)).$$

4.4 Membership

Subsets of languages only cover a subset of syntactic structures accepted by a language, where we do not consider empty sets. The relationship *subsetOf* covers this subset relationship and *subsetEqOf* additionally extends it by reflexivity.

$$subsetOf(l,l') \Rightarrow Language(l) \wedge Language(l').$$
$$subsetOf(l,l') \Rightarrow \neg subsetOf(l',l).$$
$$subsetEqOf(l,l') \Rightarrow Language(l) \wedge Language(l').$$
$$subsetEqOf(l,l') \Leftrightarrow l = l' \vee subsetOf(l,l').$$

Further, we use the types *Tuple* and *Sequence* to enable the representation of tuples and sequences of artifacts in our axiomatization, respectively. The subscript relationship at_i is used to refer to an artifact in a tuple or a sequence by an index. We assume that *Pair* is a shortcut for tuples of two artifacts.

$$Tuple(t) \Rightarrow Entity(t).$$
$$Sequence(t) \Rightarrow Entity(t).$$
$$at_i(a,t) \Rightarrow Artifact(a) \wedge (Tuple(t) \vee Sequence(t)) \text{ for } i \in \mathbb{N}.$$

[10] http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF/

Table 4: Exemplary part-hood, definition and implementation relationships for EMF.

Entities & Relationships	Comments
Technology(EMF)	Eclipse Modeling Framework
Technology(EMFCore)	Covers core functionality for Ecore etc.
partOf(EMFCore,EMF)	EMFCore is part of EMF
partOf(org.eclipse.emf.ecore,EMFCore)	The ecore package is part of EMFCore
Technology(EMFPersistence)	The technology for XMI serialization
partOf(EMFPersistence,EMF)	The persistence framework is a part of EMF
System(MyModelProject)	An eclipse EMF modeling project
partOf(MyMeta.ecore,MyModelProject)	Metamodel is part of the project
partOf(MyMeta.genmodel,MyModelProject)	Generation model is part of the project
partOf(metaname,MyMeta.ecore)	Metamodel's name is defined in the Ecore file
partOf(MyMeta.genmodel.ref,model.genmodel)	Generation model refers to the metamodel
Language(XML)	eXtensible Markup Language
Language(XMI)	XML Metadata Interchange format
Specification(XMISpec)	The XMI specification by the OMG
defines(XMISpec,XMI)	The specification gives a definition for XMI
Language(ECore.XMI)	The restricted XMI language for .ecore files
defines(MyMeta.ecore, MyModels.XMI)	Metamodel defines the language of instance models
Language(MyModels.XMI)	The restricted language for instance models in XMI
Language(JavaObjects)	Language for Java objects at runtime
Language(ECore)	The language for models in non-serialized form
Function(saveModel)	Persists a given transient model in the XMI format
Function(loadModel)	Loads a model from a given XMI file
implements(EMFCore,ECore)	Core Technology implements ECore language
implements(EMF,ECore.XMI)	EMF implements metamodel serialization
implements(EMFPersistence, saveModel)	Persistence framework realizes serialization
implements(EMFPersistence, loadModel)	Persistence framework realizes deserialization

Table 5: Exemplary relationships on *subsetOf* and *elementOf* for EMF.

Relationships	Comments
subsetOf(XMI,XML)	XMI is a subset of XML based on an OMG standard
subsetOf(Ecore.XMI, XMI)	The XMI format for models is a restricted form of XMI
subsetOf(Models.XMI, XMI)	The XMI format for instance models is a restricted form of XMI
elementOf(MyMeta.ecore,Ecore.XMI)	Metamodel is written in the restricted XMI format.
elementOf(MyModel.xmi, MyModels.XMI)	Instance model is written in restricted XMI for instances
elementOf(MyModel,JavaObjects)	The transient model is a Java object

An artifact is an element of a *Language* instance, if the artifact conforms to the defining specification. Scenarios on language membership are presented in Table 5. An artifact uses a language, if there is at least one part that is an element of the language. Usage can be similarly defined for systems and technologies.

$$\begin{aligned}&\mathit{elementOf}(x,y) \Rightarrow \mathit{Artifact}(x) \wedge \mathit{Language}(y)\\&\quad \vee \mathit{Pair}(x) \wedge \mathit{Function}(y).\\&\mathit{elementOf}(a,l) \Leftarrow \exists s.\mathit{defines}(s,l) \wedge \mathit{conformsTo}(a,s).\end{aligned}$$

4.5 Conformance Relationship

An artifact may offer a formal description of a language's syntax and semantics. We introduce *conformsTo* to cover conformance of an artifact with a syntax definition (a 'schema') where syntax should be understood here in a broad sense. For instance, we also view types, as defined by type systems, as languages.

In order to make engineers aware of the meaning of conformance, we axiomatize it as far as possible; the axiomatization is idealized and incomplete. If the schema and the instance are composite, for every part of the instance exists a schema part such that the former conforms to the latter. Otherwise the instance is not composite and it is an element of the language defined by the schema part.

Actual conformance deviates from the idealized axiomatization, if the mapping from schema composites to instance components or vice versa is not deterministic or otherwise unclear. We can further define logical equivalence for conformity informally as follows. An instance conforming to a language specification artifact implies that the schema defines the instance's language. Illustrative examples on conformance are presented in Table 6.

$$\begin{aligned}&\mathit{conformsTo}(a,d) \Rightarrow \mathit{Artifact}(a) \wedge \mathit{Artifact}(d)\\&\mathit{conformsTo}(a,a') \Leftarrow (\forall p.\mathit{partOf}(p,a) \wedge \exists p'.\mathit{partOf}(p',a')\\&\quad \wedge \mathit{conformsTo}(p,p')) \vee \exists t.\mathit{defines}(a',t) \wedge \mathit{elementOf}(a,t)\end{aligned}$$

4.6 Correspondence Relationship

Correspondence expresses that two artifacts of typically different languages represent the same data. Correspondence is necessary when talking about mapping or modeling technologies, such as Hibernate and EMF.

We axiomatize an idealized correspondence of artifacts x and y as follows. If the artifacts are composite, then for each part of x there is a corresponding part of y and vice versa. Otherwise a value level is reached and both artifacts (parts) are equal.

$$\begin{aligned}
&\mathit{correspondsTo}(x,y) \Rightarrow \mathit{Artifact}(x) \wedge \mathit{Artifact}(y)\\
&\mathit{correspondsTo}(x,y) \Leftarrow\\
&\quad(\forall px.\mathit{partOf}(px,x) \Rightarrow \exists py.\mathit{partOf}(py,y)\\
&\quad\ \wedge \mathit{correspondsTo}(px,py))\\
&\quad\wedge (\forall py.\mathit{partOf}(py,y) \Rightarrow \exists px.\mathit{partOf}(px,x)\\
&\quad\ \wedge \mathit{correspondsTo}(py,px))\\
&\quad\vee (\nexists p.\mathit{partOf}(p,x) \vee \mathit{partOf}(p,y)) \wedge \mathit{sameAs}(x,y)
\end{aligned}$$

The axiomatization above does not consider issues due to even simple forms of 'impedance mismatch'. For instance, an artifacts may have parts that do not correspond to any part on the other side. An artifact may also have a level of composition that is not present on the other side. Correspondence scenarios can be found in Table 6.

4.7 Traceability

A trace is a nested sequence that refers to pairs of artifacts. For simplicity, we assume here that traces contain artifacts (parts thereof) rather than 'references'. Traces help comprehending relationships between artifacts.

$$\begin{aligned}
&\mathit{Trace}(t) \Rightarrow \mathit{Sequence}(t)\\
&\mathit{Trace}(t) \Rightarrow \forall i.\exists p.\mathit{at}_i(p,t)\\
&\quad\Rightarrow \mathit{Pair}(p) \wedge (\exists a1,a2.\mathit{at}_1(a1,p) \wedge \mathit{at}_2(a2,p)\\
&\quad\ \wedge \mathit{Artifact}(a1) \wedge \mathit{Artifact}(a2))
\end{aligned}$$

For instance, a trace may represent 'evidence' for conformance or correspondence relationships by pairing up related parts (fragments).

$$\begin{aligned}
&\mathit{traceOf}(t,aa) \Rightarrow \mathit{Pair}(aa) \wedge \exists a1,a2.\mathit{Artifact}(a1)\\
&\quad\wedge \mathit{Artifact}(a2) \wedge \mathit{at}_1(a1,aa) \wedge \mathit{at}_2(a2,aa) \wedge \mathit{Trace}(t)\\
&\mathit{traceOf}(t,aa) \Rightarrow \exists a1,a2.\mathit{at}_1(a1,aa) \wedge \mathit{at}_2(a2,aa)\\
&\quad\wedge (\forall i.\exists p.\mathit{at}_i(p,t) \Rightarrow\\
&\quad\ (\exists b1,b2.\mathit{partOf}(b1,a1) \wedge \mathit{partOf}(b2,a2)\\
&\quad\ \wedge \mathit{at}_1(b1,p) \wedge \mathit{at}_2(b2,p)))
\end{aligned}$$

4.8 Functions and Applications Thereof

We abstract from implemented operations in terms of mathematical functions, whose domain and range are languages. A technology implements functions that can be reused in many contexts. For example, the Java Architecture for XML Binding (JAXB) [11] offers support for these major operations: Converting XML content into a Java representation and vice versa and further support for accessing and updating the Java representation of the XML based content.

$$\begin{aligned}
&\mathit{Function}(f) \Rightarrow \exists d,r.\mathit{rangeOf}(r,f) \wedge \mathit{domainOf}(d,f)\\
&\mathit{domainOf}(d,f) \Rightarrow \mathit{Language}(d) \wedge \mathit{Function}(f)\\
&\mathit{rangeOf}(r,f) \Rightarrow \mathit{Language}(r) \wedge \mathit{Function}(f)
\end{aligned}$$

A function is applied with a pair that consists of an actual input and output. An input of an application pair is valid, if it is an element of the function's domain. Further, the pair's output is valid, if it is an element of the function's range.

$$\begin{aligned}
&\mathit{inputOf}(i,p) \Rightarrow \mathit{Artifact}(i) \wedge \mathit{Pair}(p)\\
&\mathit{inputOf}(i,p) \Rightarrow \exists d,f.\mathit{Language}(d) \wedge \mathit{Function}(f)\\
&\quad\wedge \mathit{at}_1(i,p) \wedge \mathit{domainOf}(d,f) \wedge \mathit{elementOf}(i,d)\\
&\quad\wedge \mathit{elementOf}(p,f)
\end{aligned}$$

$$\begin{aligned}
&\mathit{outputOf}(o,p) \Rightarrow \mathit{Artifact}(o) \wedge \mathit{Pair}(p)\\
&\mathit{outputOf}(o,p) \Rightarrow \exists r,f.\mathit{Language}(r) \wedge \mathit{Function}(f)\\
&\quad\wedge \mathit{at}_2(o,p) \wedge \mathit{rangeOf}(r,f) \wedge \mathit{elementOf}(o,r)\\
&\quad\wedge \mathit{elementOf}(p,f)
\end{aligned}$$

At this point, we can clarify that artifacts do not only manifest as files, folders, and fragments. The type *Transient* proxies for artifacts that occur as intermediate results—possibly 'computed' by functions. Examples for transients and function application scenarios are given in Table 7.

$$\begin{aligned}
&\mathit{Transient}(t) \Rightarrow \mathit{Artifact}(t)\\
&\mathit{Transient}(t) \Rightarrow \exists f,p.\mathit{Function}(f) \wedge \mathit{Pair}(p)\\
&\quad\wedge \mathit{elementOf}(p,f) \wedge \mathit{outputOf}(t,p)
\end{aligned}$$

Based on prior definitions, we extend the axiomatization for language membership as follows. An artifact is an element of a language, if there exists a technology, which implements the language and provides a function that accepts the artifact.

$$\begin{aligned}
&\mathit{elementOf}(p,f) \Rightarrow \exists d,r,i,o.\mathit{domainOf}(d,f) \wedge \mathit{rangeOf}(r,f)\\
&\quad\wedge \mathit{inputOf}(i,p) \wedge \mathit{outputOf}(o,p) \wedge \mathit{elementOf}(i,d)\\
&\quad\wedge \mathit{elementOf}(o,r).\\
&\mathit{elementOf}(a,l) \Leftarrow \exists t.\mathit{Technology}(t) \wedge \mathit{implements}(t,l)\\
&\quad\wedge \exists f.\mathit{Function}(f) \wedge \mathit{implements}(t,f)\\
&\quad\wedge \exists p.\mathit{Pair}(p) \wedge \mathit{elementOf}(p,f) \wedge \mathit{inputOf}(a,p)
\end{aligned}$$

[11] https://jaxb.java.net/

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

Table 6: Exemplary correspondence and conformance relationships for EMF.

Relationships	Comments
correspondsTo(MyModel,MyModel.xmi)	The Java object corresponds to the XMI serialized format
conformsTo(MyModel,MyMeta.ecore)	The transient instance model conforms to the ecore model
conformsTo(MyModel.xmi, MyMeta.ecore)	The XMI instance model conforms to the ecore model

Table 7: Exemplary scenarios on function application for EMF.

Relationships	Comments
domainOf(JavaObjects,saveModel)	saveModel takes an instance model as a Java object as input
rangeOf(MyModels.XMI,saveModel)	saveModel XMI files conform to the Ecore metamodel as output
Pair(SavePair)	An input-output pair for saving a model
elementOf(SavePair,saveModel)	The IO pair is a member of the save function
inputOf(MyModel,SavePair)	The transient is the input of the pair
outputOf(MyModel.xmi,SavePair)	The XMI file is the output

4.9 Facilitation and Usage

We describe another basic subtype, *Concept*, which deals with conceptual entities at a higher level of abstraction, e.g., design patterns or protocols. These concepts are practices that are defined in artifacts (i.e., more or less formal specifications). Thus, concepts are a bit like languages. An entity may be said to conform to a concept, if it correctly follows the concept's definition. We require that one can derive a predicate from a concept's definition to decide on such conformance.

$$Concept(c) \Rightarrow Entity(c)$$
$$Concept(c) \Rightarrow \exists a.Artifact(a) \land defines(a,c)$$

Next, we introduce the usage relationship which is related to reuse of technologies and architectures in software engineering. A technology, system or artifact are used as soon as they are referred to. The using types are technology, system, and artifact except that typically we do not expect a technology to use a system.

A technology, system or artifact can use a language. Then, at least a part of the using side has to be written in this language. Concept usage is similar to language membership on a more abstract level. A system can use a concept that is defined by an artifact, e.g., a design pattern, if some parts of it conform to the concept, as discussed above. We specify the types admitted to *uses* relationships in Table 8.

$$uses(x,y) \Leftarrow \exists p.partOf(p,x) \land elementOf(p,y)$$
$$uses(x,y) \Leftarrow \exists s,p.defines(s,y) \land partOf(p,x) \land conformsTo(p,s)$$

A technology facilitates the compliance with a concept, such as MVC or Restful services, by providing means to use it. Thus, facilitation implies a deferred usage. If a system uses the technology in the correct way, then it also uses the concept. Examples of facilitation and usage can be found in Table 9.

Table 8: Types involved in *uses* relations, where the left side is the user.

	Language	**Technology**	**System**	**Artifact**	**Concept**
Language					x
Technology	x	x		x	x
System	x	x	x	x	x
Artifact	x	x	x	x	x
Concept					x

$$facilitates(x,y) \Rightarrow Technology(x) \land Concept(y)$$
$$facilitates(x,y) \Rightarrow \forall s.System(s) \land (uses(s,x) \Rightarrow uses(s,y))$$

5 CONCLUSION

According to Smith et al. (Smith and Welty, 2001), database and information systems, software engineering (in particular, domain engineering), and artificial intelligence create a demand for the application of ontologies in computer science. In this paper, we are concerned with software engineering—not with domain engineering, but with software technologies used in software development or software systems.

The types of entities and relationships, as axiomatized in this paper, are useful in authoring metadata that describes software technologies in a manner that software developers may better understand how to use the technologies and the languages that go with them. Megamodels written in a language like MegaL (Favre et al., 2012a) describe specific usage of technologies in specific systems or patterns thereof. The emerging ontology SoLaSoTe serves as a knowledge base to gather general facts from these models and to integrate (to collect) megamodels.

SoLaSoTe complements other ontologies in the software engineering field. For instance, Solanki et al. (Solanki et al., 2016) introduce the suite of ontolo-

Table 9: Exemplary concepts, facilitation and usage relationships for EMF.

Entities & Relationships	Comments
Concept(XMIserialization)	XMI serialization is a concept
Concept(SoftwareModeling)	Software Modeling is a concept
defines(XMISpec,XMIserialization)	The specification describes how to serialize objects in XMI
uses(MyModelsProject,XMIserialization)	The project can use the concept of XMI serialization
uses(MyModelsProject,SoftwareModeling)	The project contains actual models
facilitates(EMFPersistence, XMIserialization)	The EMF persistence framework supports the serialization
facilitates(EMF,SoftwareModeling)	EMF supports developers to model software

gies *ALIGNED* connecting the fields of software and data engineering. This suite contains software engineering-specific knowledge, e.g., on the software lifecycle[12] that is regarded as a process and defined by its activities. Oberle et al. (Oberle et al., 2004; Oberle et al., 2006) discuss the definition of general software ontologies resulting in a 'Core Software Ontology', a 'Core Ontology for Software Components' and a 'Core Ontology of Services'[13].

Clearly, different ontologies may exist for one domain (Corcho et al., 2006). The kind of knowledge and the ontology's structure depend on the point of view. Based on our previous research we had a certain scope in mind and we matured our point of view by a literature survey. A more extensive study possibly with additional research questions is, of course, an interesting direction for future work.

In developing the emerging SoLaSoTe ontology, we aim at applying relevant best practices or quality criteria for ontologies (Corcho et al., 2006). For each concept, there is informal text meant to make the axioms more understandable and to motivate the concepts and their relationships. Extensibility is provided because new subtypes of given concepts and more refined relationships can be introduced. We covered technologies across three different technological spaces (only EMF is shown in this paper). This makes us assume that the axiomatization is coherent.

The axiomatization mainly serves as a reference schema and 'semantics' for a vocabulary to be used in technology documentation. The axioms describe general properties of relationships; they are 'blueprints' for interpretations (Lämmel and Varanovich, 2014) (in fact, software analyses) that implement relationships for specific technologies modeled by specific megamodels. A limited version of the ontology is in use in the semantic wiki of '101' (Favre et al., 2012b).

REFERENCES

Bézivin, J., Jouault, F., and Valduriez, P. (2004). On the Need for Megamodels. In *Proc. OOPSLA/GPCE: Best Practices for Model-Driven Software Development workshop*.

Calero, C., Ruiz, F., and Piattini, M., editors (2006). *Ontologies for Software Engineering and Software Technology*. Springer.

Corcho, O., Fernández-López, M., and Gómez-Pérez, A. (2006). Ontological engineering: principles, methods, tools and languages. In *Ontologies for software engineering and software technology*, pages 1–48. Springer.

d'Aquin, M. and Gangemi, A. (2011). Is there beauty in ontologies? *Applied Ontology*, 6(3):165–175.

Diskin, Z., Kokaly, S., and Maibaum, T. (2013). Mapping-Aware Megamodeling: Design Patterns and Laws. volume 8225 of *LNCS*, pages 322–343. Springer.

Elberzhager, F., Münch, J., and Nha, V. T. N. (2012). A systematic mapping study on the combination of static and dynamic quality assurance techniques. *Information & Software Technology*, 54(1):1–15.

Favre, J., Lämmel, R., and Varanovich, A. (2012a). Modeling the Linguistic Architecture of Software Products. In *Proc. MODELS 2012*, volume 7590 of *LNCS*, pages 151–167. Springer.

Favre, J.-M., Lämmel, R., Schmorleiz, T., and Varanovich, A. (2012b). 101companies: A Community Project on Software Technologies and Software Languages. In *Proc. TOOLS 2012*, volume 7304 of *LNCS*, pages 58–74. Springer.

Gangemi, A., Guarino, N., Masolo, C., Oltramari, A., and Schneider, L. (2002). Sweetening Ontologies with DOLCE. In *Proc. EKAW 2002*, volume 2473 of *LNCS*, pages 166–181. Springer.

Kurtev, I., Bézivin, J., and Akşit, M. (2002). Technological Spaces: an Initial Appraisal. In *Proc. of CoopIS, DOA 2002, Industrial track*.

Lämmel, R. and Varanovich, A. (2014). Interpretation of Linguistic Architecture. In *Proc. ECMFA 2014*, volume 8569 of *LNCS*, pages 67–82. Springer.

Oberle, D., Eberhart, A., Staab, S., and Volz, R. (2004). Developing and Managing Software Components in an Ontology-Based Application Server. In *Proc. Middleware 2004*, volume 3231 of *LNCS*, pages 459–477. Springer.

Oberle, D., Lamparter, S., Grimm, S., Vrandecic, D., Staab, S., and Gangemi, A. (2006). Towards ontologies for formalizing modularization and communication in large software systems. *Applied Ontology*, 1(2):163–202.

Ruiz, F. and Hilera, J. R. (2006). Using Ontologies in Software Engineering and Technology. In *Ontologies*

[12] http://aligned.cs.ox.ac.uk/ont/slo.html

[13] http://km.aifb.kit.edu/sites/cos/

for Software Engineering and Software Technology, pages 49–102. Springer.

Smith, B. and Welty, C. A. (2001). FOIS introduction: Ontology - towards a new synthesis. In *FOIS*, pages iii–ix.

Solanki, M., Božić, B., Freudenberg, M., Kontokostas, D., Dirschl, C., and Brennan, R. (2016). *Enabling Combined Software and Data Engineering at Web-Scale: The ALIGNED Suite of Ontologies*, pages 195–203. Springer, Cham.

Tran, T. and Debruyne, C. (2012). Towards Using OWL Integrity Constraints in Ontology Engineering. In *Proc. OTM 2012*, volume 7567 of *LNCS*, pages 282–285. Springer.

Wohlin, C. (2014). Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proc. EASE 2014*, pages 38:1–38:10.