An Ontology-based Approach for Domain-driven Design of Microservice Architectures

Andreas Diepenbrock[1], Florian Rademacher[2] and Sabine Sachweh[3]

Abstract: Microservice Architecture (MSA) is an emerging service-based architectural style that focuses on the design and implementation of highly scalable distributed software systems. To analyze the business domain and its decomposition into services Domain-driven Design (DDD) is commonly applied. DDD is an approach for designing software that relies on various model-based concepts to express knowledge about the business domain, e.g. the Bounded Context (BC) pattern, which clusters a set of coherent Domain Models (DM). In addition to the fact that MSAs fosters a high degree of team independence, the uncoordinated evolution of DMs that originally were semantically equivalent or partially shared similar semantics can occur. In this paper, we identify challenges related to the semantic decoupling of DMs. Additionally, we present a metamodel for modeling MSAs based on the principles of DDD which allows the expression of semantics for relationships between fragmented DMs and SMs.

Keywords: Ontologies, Domain-driven Design, Microservices, Web Ontology Language

1 Introduction

With the increasing need of highly scalable and distributed systems, the Microservice Architecture (MSA) style has emerged [Ne15]. MSA focuses on decomposing a software architecture into services that are responsible for providing cohesive business or technological capabilities distinct from other services and fosters a high degree of team independence [RSZ17, VC16]. MSA arises from the Service Oriented Architecture (SOA) style which also focuses on the decomposition a software architecture into services, but requires an Enterprise-Service-Bus (ESB) for routing and sending messages among services. While the philosophy behind SOA can be described as "share-as-much-as-you-can" the MSA philosophy is "share-nothing", which fosters the usage of agile development methods,

[1] University of Applied Sciences and Arts Dortmund, Institute for Digital Transformation of Application and Living Domains (IDiAL), Otto-Hahn-Straße 23, 44227 Dortmund, andreas.diepenbrock@fh-dortmund.de
[2] University of Applied Sciences and Arts Dortmund, Institute for Digital Transformation of Application and Living Domains (IDiAL), Otto-Hahn-Straße 23, 44227 Dortmund, florian.rademacher@fh-dortmund.de
[3] University of Applied Sciences and Arts Dortmund, Institute for Digital Transformation of Application and Living Domains (IDiAL), Otto-Hahn-Straße 23, 44227 Dortmund, sabine.sachweh@fh-dortmund.de

doi:10.18420/in2017_177

and promotes autonomy and isolation of development teams [FLM17, YSS17]. To design systems that are based on the MSA, the Domain-driven Design (DDD) [Ev04] is commonly used [PJ16]. DDD is a model-based approach for designing complex software with evolving models representing core domain concepts and propose the use of *Domain Models* (DMs), *Bounded Context* (BC), and *Shared Models* (SMs) (cf. Section 2).

However, BCs and DMs lack support for the expression of semantics. In particular, this becomes problematic when DDD is used for MSA design, as DMs are fragmented over the system's architecture and, due to MSA's emphasis on team independence [YSS17], might be evolved independently, yet they partially share the same semantics or are semantically equivalent. Additionally, the typically high degree of independence among MSA team can lead to a different understanding of domain concepts. Furthermore, having a shared understanding of a certain domain is not only challenging when using MSA, but in software development in general [Ev04].

Ontologies are an approach for expressing knowledge and semantics in a particular domain. This knowledge can be used to formally modeling the structure of a system like entities or relationships [SS09]. Instead of classes, attributes and associations, ontologies leverage concepts, properties and relations to express knowledge about the domain and domain-related coherences. These concepts are used to express knowledge in a form that allows automatic interpretation and derivation of additional semantical knowledge about the specific domain. For the expression of ontologies, several languages have been proposed, with the Web Ontology Language (OWL) being one of the most popular ones besides the Resource Description Framework (RDF) [BPD09].

In this paper we address DDD's lack of support for expressing semantics in DDD-based models by means of ontologies and OWL. Thereby, our contribution is twofold. First, to enable the expression of semantic relationships in a domain-driven MSA design, we define a metamodel for DDD that comprises concepts for DMs, their assignment to BCs and the definition of SMs between BCs. Second, we extend this metamodel with semantical capabilities that enable the expression of semantics of DMs and SMs and their relationships across BCs. The application of our metamodel allows (i) to use DDD for modeling BCs, DMs and SMs and the foundation of business related Microservices; (ii) enrich DMs and SMs with semantic to ensure a common understanding of domain concepts in distributed MSA development.

The remainder of this paper is structured as follows. In Section 2 we describe the design of MSAs based on the DDD approach. Section 3 introduce a running example that is used for the identification of challenges of fragmented Domain Models in an MSA which is described in Section 4. Based on these challenges, Section 5 presents our metamodel for modeling an MSA following the DDD approach and enrich with semantical knowledge. Section 6 presents related work in this field and Section 7 concludes and describes future work.

2 Domain-driven Design of Microservice Architectures

When designing an MSA, a number of technical and organizational aspects have to be considered. While in the monolithic architecture style the complete application shares the same codebase, it is split into multiple independent Microservices in an MSA [Na16, Vi15]. The independence of MSA also has an impact on the team independence, also MSA fosters benefits like (i) flexible adaption of software systems; (ii) increased software quality because of isolated testability, higher resilience and runtime scalability; (iii) increased development team productivity as the architecture's structure can be aligned with the team structure and each service being maintained by exactly on team [RSZ17].

A well established approach for decomposing a software into Microservices is DDD [Na16]. It propose to use DMs for expressing domain concepts and the application of the BC pattern to cluster coherent DMs and express the boundaries of Microservices [Wo16]. A DM represents a domain concept and its characteristics, which are modeled as attributes of a DM. However, in distributed software systems like MSAs there might be many DMs and DDD proposes to cluster related DMs in BCs. The concept of BCs is used for modeling the domain concepts encapsulated by a business-related Microservice [Ne15]. Thereby, sharing relationships between BCs have to be made explicit in the form of SMs. An SM is a tailored version of a DM and contains only those domain attributes that need to be shared with another BC.

Figure 1 shows DDD modeling concepts relevant to domain-driven MSA design. Table 1 explains each concept depicted in Figure 1.

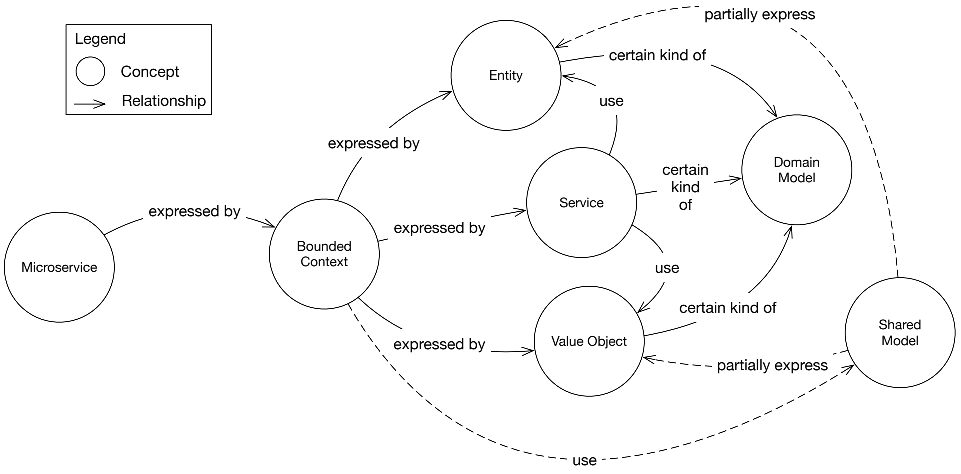

Fig. 1: Navigation Map for Domain-driven Design in MSA

When applying DDD as well as DMs, SMs and BCs for decomposing a certain domain, each of the mentioned modeling patterns can be mapped to an MSA component [Wo16].

Concept	Description
Bounded Context	A Bounded Context depicts the boundaries of a coherent part of the business domain. BCs can be nested, to express distinct sub-contexts related to the modeled BC. The concept of BCs is used for modeling the domain concepts encapsulated by a business-related Microservice.
Entity	Entities represent domain concepts with an identity and encapsulates related domain characteristics in the form of attributes. When changing attribute values of an entity, the identity of the entity itself is not affected.
Value Object	Like Entities, Value Objects represent domain concepts and encapsulate certain domain characteristics in the form of attributes. However, they do not exhibit a domain-oriented identity.
Service	Services represent functionality relevant to the modeled domain and differs to Microservices in form of internal functionality, e.g. Data Access Objects, Repositories for Entities or Value Objects or prepare and processing data. Services are only for internal use and are not exposed to other Microservices.
Domain Model	Domain Models are proposed for expressing domain concepts related to BCs. Furthermore, Entities, Services and Value Objects are certain kinds of DMs and express the internal domain concepts of the related BC.
Shared Model	Like Domain Models, Shared Models are also proposed for expressing domain concepts but related to other Microservices. With SMs also the partial expression of relevant domain concepts and their associated attributes are possible.

Tab. 1: Explanation of relevant concepts for the Domain-driven Design of MSAs [Ev04]

Each Microservice typically realizes a given BC and the DMs assigned to it. The SMs of the BC define the foundation for the respective Microservice's interface, whose purpose is to expose instances of the SMs to other services or external callers.

When designing software following the principles of DDD, the language of domain experts is used for naming of classes, attributes and relationships [Ev04]. DDD fosters the formation of a *Ubiquitous Language* (UL) that represents a consistent, unambiguous terminology of the respective domain and is applied in discussions between domain experts and developers as well as in all DDD-based models and related software implementations.

3 Running Example

Figure 2 shows the DDD-based model of an application called Research Management System (RMS), which we use in our research group to manage conference calls and papers. We leveraged packages and classes from UML to depict BCs, DMs and SMs. The domain-related structure of DMs or SMs and association between these are expressed by UML attributes and associations.

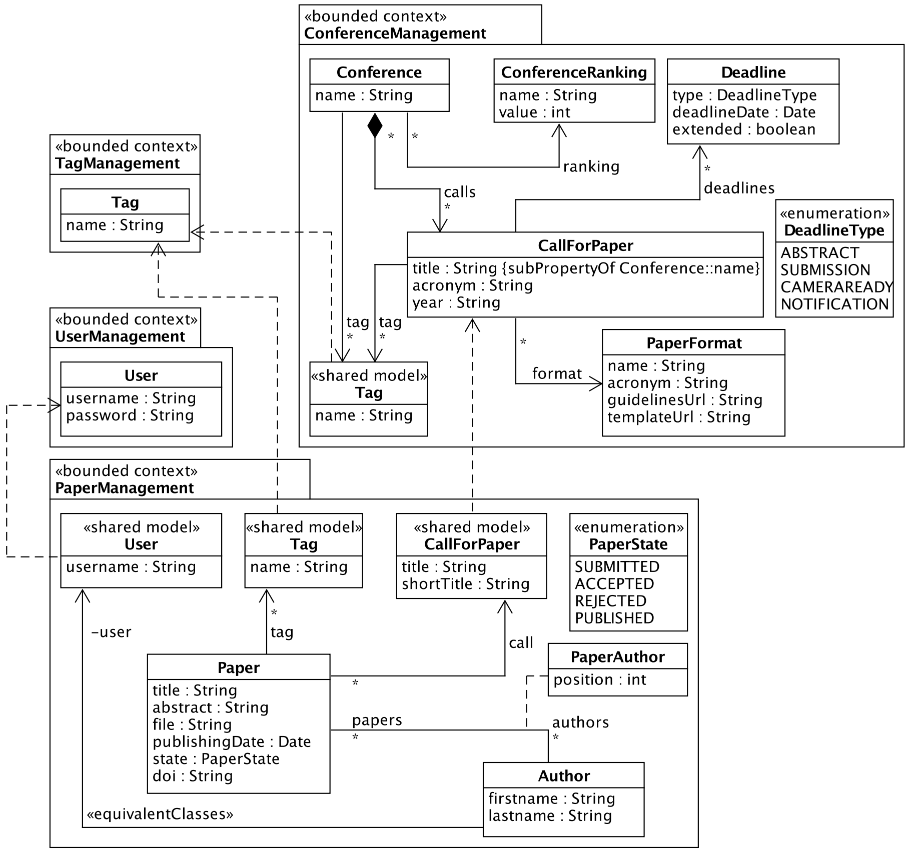

Fig. 2: DDD-based running example

The RMS allows to manage conferences, corresponding call for papers and written papers. Furthermore, information about the required paper format and important deadlines are maintained. To simplify the search for a suitable conference, calls for papers and written papers can be tagged. Papers may have several authors whose position on the paper can be configured. Additionally, a paper might be in one of the statuses “submitted”, “accepted”, “rejected” or “published”. Also an state for papers can be managed e.g. to distinguish between submitted and accepted papers.

The domain-driven design is composed of the BCs `TagManagement`, `UserManagement`, `PaperManagement` and `ConferenceManagement`. To manage the corresponding Tags of `Conference`, `CallForPaper` and `Paper` may have several Tags assigned. The `Tag` DM is part of the `TagManagement` BC. A `Paper` of the BC `PaperManagement` is assigned to a

CallForPaper of the ConferenceManagement BC, also the Paper may have multiple authors that are represented by Author in PaperManagement. Managing a User is settled in the BC UserManagement for authentication purpose.

Because the BC PaperManagement as well as the BC ConferenceManagement have to interact with other Microservices, these have defined Shared Model elements which are expressed in Figure 2 as classes with the stereotype <<shared model>>. In PaperManagement the SM User is defined that relates to the DM User of the UserManagement BC. This allows the relationship of Paper instances through the Author with User. Also the SM Tag in BC PaperManagement is defined that represents the DM Tag of the TagManagement BC. The relationship of DM Paper with the CallForPaper is realized by the SM CallForPaper within the PaperManagement BC. Realization of the SM User as well as CallForPaper within the BC PaperManagement are different than the respective DMs they relate to. Also the BC ConferenceManagement defines an SM Tag which relates to the DM Tag of the BC TagManagement. The relationship in Figure 2 between SM and the corresponding DM are visualized by the dashed line and the arrow is pointing to the respective DM.

4 Challenges of fragmented Domain Models in MSA

With fragmented DMs in a DDD-based MSA design (i) syntactical; (ii) structural; (iii) semantical challenges may occur [SvH06]. Syntactical challenges result from application of heterogeneous data formats, e.g. HTML, XML, JSON, RDF when data is exchanged and transformations between them can be problematic [SvH06]. Also the scope of these data formats can be different. While HTML, XML and JSON mainly used for expressing data, RDF is commonly used for defining data and its meaning in a meta-level. Structural challenges comprise the sub-challenges *bilateral conflicts*, *multilateral conflicts* and *meta-level conflicts*. While bilateral conflicts involve one element found in different sources, multilateral conflicts involve more than one element. Meta-level conflicts occur when using different modeling elements to represent the same kind of information. Semantical challenges results from a different understanding or mental representation of the domain. Among others, this may result in different naming schemes for the same concepts.

4.1 Semantical Challenge: Derived Domain Model Attributes

A typical semantical challenge occurs when attributes of domain concepts can be derived from other attributes. For example a Microservice that realizes the ConferenceManagement BC might expose concrete instances of the CallForPaper concept via its interfaces. However, due to convenience reasons, the developer team responsible for the service, might decide to expose these instances as string in which the acronym and year attributes are concatenated, e.g. a CallForPaper instance cfp with cfp.acronym = "BDSDST" and cfp.year = 2017 results in an exposed instance cfp = "BDSDST2017". Thus, the MSA must apply a merge transformation on the received data to get the merged string.

Hence, attribute values might be derived from other attribute values, which poses a semantical challenge because the data might be fragmented over a distributed software design like MSA, because the underlying concepts are also fragmented.

4.2 Different Understanding of Shared Domain Concepts

As described in Section 2, when designing an MSA with DDD, a common understanding of domain concepts is required. Otherwise different mental representations of domain concepts, e.g. divergent names of semantically equivalent concepts or attributes, might exist. In [Ev04] Evans emphasizes the importance of an explicit *Ubiquitous Domain Language* to prevent different understandings of shared concepts. However, DDD lacks support for formally expressing and maintaining such a language.

For example in Figure 2 the BCs `ConferenceManagement` and `PaperManagement` might have a different mental model for expressing a headline. In `ConferenceManagement` the attribute `name` contains a headline, while in `PaperManagement` the attribute `title` represents a headline. The risk of diverging understandings of semantically equivalent domain concepts increases when begin to implement their Microservices. As MSA explicitly fosters a high degree of team independence, especially regarding implementation decisions, the occurrence of different representations of equivalent concepts becomes more likely over time.

4.3 Distributed semantically equivalent Domain Concepts

Monolithic applications share the same codebase, which results in complex domain models. This results in an increased complexity of domain models as a monolith typically represents a coherent, rather large application, which then is also the case for its underlying domain representation. While MSAs are explicitly decomposed in specialized, distinct domain models following a “divide-and-conquer approach” [Na16]. When designing an MSA these DMs can be associated across multiple independent Microservices with SMs.

In Figure 2 the BC `PaperManagement` has defined an SM `User`, which relates to the DM `User` of the BC `UserManagement`. The DM `Author` of `PaperManagement` is relating to the DM `User` of the BC `UserManagement` through the SM `User` of the BC `PaperManagement`. When sharing data about the user between these Microservices a common approach must be used to identify the same object, e.g. a comprehensive mechanism for generating identifiers must be defined across teams.

5 Integrating Ontologies in Bounded Contexts

In the following we address the three challenges described in Subsection 4.1, 4.2 and 4.3 with a model-based approach that relies on a metamodel, which combines elements of DDD and OWL-based ontology modeling.

A metamodel represents the abstract syntax of a formal modeling language, i. e. it defines the concepts supported by the language and their relationships [FB05]. This abstract syntax provides the basis for a concrete language syntax. However, the definition of a concrete language syntax is beyond the scope of this paper.

The following subsection presents the DDD-related concepts of the metamodel, which is enriched with semantic concepts leveraging the *Conceptual Data Modeling* (CDM) approach in Subsection 5.2.

5.1 A Metamodel for Domain-driven Design of Microservice Architectures

Starting from the domain model depicted in Figure 2 we identify concepts that have to be part of the metamodel for DDD, which is shown in Figure 3. Due to the aspect of team independence, which is an important characteristic of MSA development, our metamodel is intended to model a Microservice instead of the entire software architecture.

As described in Section 2 a BC can be mapped to a specific Microservice. However, it is likely that the modeled Microservice has to interact with other Microservices leveraging certain SMs from these services. Therefore, in our metamodel we distinguish between `ExternalContexts` and `BoundedContexts`. To use DMs that are maintained by other Microservices, these must be modeled as `SharedModels` inside an `ExternalContext`. This allows to define tailored versions of DMs that only comprise those attributes relevant for sharing with other services. For modeling the Bounded Context of the respective Microservice our metamodel comprises the `BoundedContext` concept. `ExternalContext` and `BoundedContext` are derived from `AbstractContext`. Hence both have a `name` for the purpose of distinction from other contexts. Contexts can be nested via the `nestedContexts` association of `AbstractContext`.

For modeling domain concepts we define the `SharedModel` and `DomainModel` concepts in our metamodel. Both concepts inherit from `AbstractModel` and thus contain a `name` attribute for referencing purposes. An `ExternalContext` must comprise at least on `SharedModel`, while a `BoundedContext` has to be associated with at least one `DomainModel`. A `DomainModel` can be defined as abstract, i.e. it might not be instantiated. Inheritance between `DomainModels` is further possible by the parent association.

Furthermore, `SharedModel` and `DomainModel` may have attributes that describe their domain-specific structures and are represented by the `Attribute` concept. An `Attribute` may have a `PrimitiveDataType` assigned to determine its datatype. These attributes are limited to `PrimitiveDataTypes` because relationships between models are expressed via instances of the `Association` concept. Associations can only have a unidirectional relationship to `SharedModel` and `DomainModel` concepts and the multiplicity is expressed by the `min` and `max` attributes of `Association`.

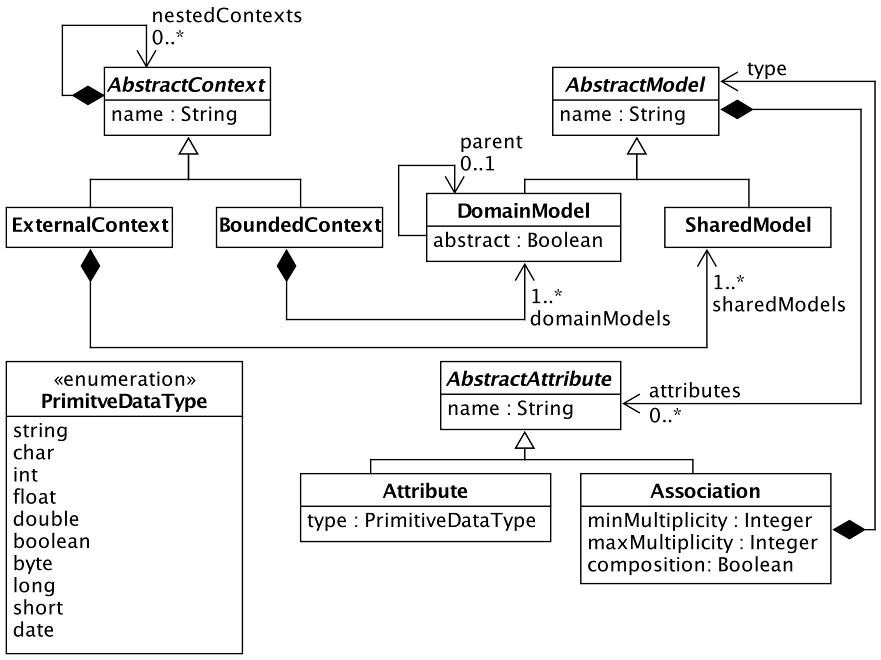

Fig. 3: Metamodel for DDD-based modeling

5.2 Enriching the Metamodel with semantic Capabilities

While the metamodel presented in Subsection 5.1 is designed for modeling an MSA following the principles of DDD, it lacks support for expressing semantical knowledge. To enrich the metamodel with semantic capabilities we distinguished between semantical characteristics of concepts related to models, i.e. `DomainModel` etc., and concepts related to attributes, i.e. `Attribute` etc. models and attributes. Semantic capabilities for model-related concepts are (i) expression of equivalence of distinct models; (ii) disjointness of distinct models; (iii) inheritance of models. For attribute-related the capabilities are composed of (i) inheritance of attributes; (ii) equivalence between distinct attributes.

For expressing semantical knowledge in our metamodel we extended it with a new abstract concept called `DomainConcept` as shown in Figure 4. Concrete instances of `DomainConcept` are `ModelConcept` and `AttributeConcept` which differentiate between semantical concepts of models and attributes to address the model-related as well as attribute-related semantical capabilities. A `DomainConcept` is expressed by a `name`. `ModelConcept` has a relation to `ModelConceptType` as well as `AttributeConcept` relates to a `AttributeConceptType`. With these two ConceptType enumerations the semantical characteristic can be expressed. The

ModelConcept relates to exactly two AbstractModel instances with the models relationship. For semantical expression of attributes the AttributeConcept is used, which relates to at least two instances of AbstractAttribute. Using the DomainConcept and the concrete instances ModelConcept and AttributeConcept allow the expression of semantical knowledge.

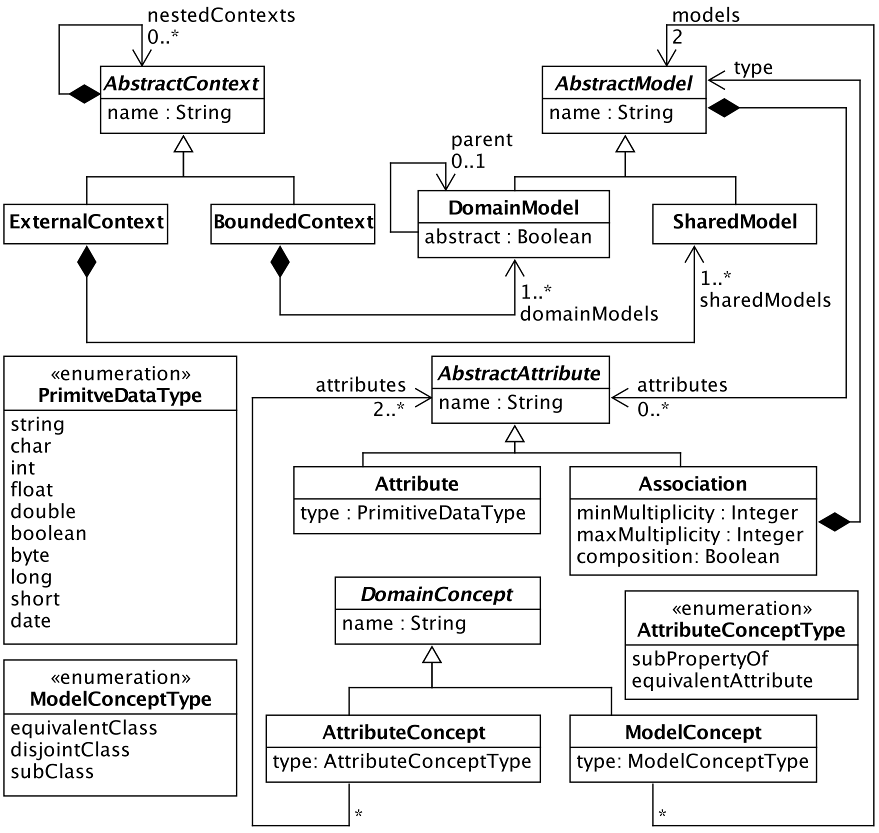

Fig. 4: Metamodel enriched with semantic capabilities through DomainConcept

We use the functional-style syntax for expressing ontological concepts in this paper, because it is given as a formal grammar and for describing OWL concepts in W3C Specification [MPSP12]. For expressing semantical knowledge in OWL a set of *axioms*[4] are defined for concepts and properties. In our metamodel these axioms are represented by ModelConceptType and AttributeConceptType, therefore each ModelConcept and AttributeConcept must have a type associated with ModelConceptType or AttributeCon-

[4] https://www.w3.org/TR/2012/REC-owl2-syntax-20121211/#Axioms

`ceptType`. Based on this `type` the relevant OWL-axiom can be derived and used for the semantical representation.

Enriching our DDD-based RMS presented in Figure 2 with semantics following the challenges presented in Section 4. While the challenge described in Subsection 4.1 by business logic through a DDD-based service (cf. Section 2), the challenges presented in Subsection 4.2 and 4.3 can be expressed by `DomainConcept` instances. Subsection 4.2 can be semantical expressed by an `AttributeConcept` for the attributes `name` and `title` of `Conference` and `CallForPaper` from BC `ConferenceManagement` with type `subPropertyOf` is modeled. The resulting semantical representation of this challenge in our running example is presented in Listing 2. While the semantical expression that the attributes `title` of `CallForPaper` and `name` of `Conference` are equivalent is expressed with `SubPropertyOf( :title :name)`, the declaration of classes and properties are derived from the modeled BC. The challenge presented in Subsection 4.3 is expressed as a `ModelConcept` with type `equivalentClass`, this represents that the SM `User` and DM `User` of BC `PaperManagement` and `UserManagement` are equivalent. Therefore, in Listing 2 the corresponding OWL syntax is presented.

```
Declaration( Class( :Conference ) )
Declaration( Class( :CallForPaper ) )
DisjointClasses( :Conference :CallForPaper )

Declaration( DataProperty ( :name ) )
DataPropertyDomain( :name :Conference )
DataPropertyRange( :name xsd:String )

Declaration( DataProperty ( :title ) )
DataPropertyDomain( :title :CallForPaper )
DataPropertyRange( :title xsd:String )

SubPropertyOf( :title :name )
```

List. 1: Expressing equivalent attributes for `ConferenceManagement` BC

```
Declaration( Class( :User ) )
Declaration( Class( :SM_User ) )
Declaration( Class( :Author ) )
SubClassOf( :User :SM_User )

EquivalentClasses( :SM_User :Author )
```

List. 2: Expressing relationships between classes for `PaperManagement` BC

With this semantic representation being derived from the modeled MSA leveraging our metamodel, knowledge about the architecture automatically be deduced by reasoners. This becomes especially useful when exchanging data between different Microservices, e.g. for integrating data with different structure or semantic and without knowledge about the underlying domain-specific concepts.

6 Related Work

To the best of our knowledge no related work for the semantical expression of distributed domain models in the context of MSA exists. Therefore, we divided the related work in the fields of applying ontologies for expressing conceptual models (cf. Subsection 6.1) and modeling Microservice Architectures with DDD (cf. Subsection 6.2).

6.1 Ontologies for expressing Conceptual Models

In [ZD07], an overview of approaches for database-centric Enterprise Integration (EI). Based on the challenge of different mental representations of conceptual models a new approach called *Semantic Integration Reflecting User-specific semantic Perspectives* (SIRUP) is presented. SIRUP addresses the problem of how individual mental domain models can be effectively preintegrated on a conceptual level. Because SIRUP addresses a database-centric scope for integrating data it can not be mapped to software architectures like MSA.

In [Ca09], Calvanese et al. propose an approach for conceptual modeling of data integration called DL-Lite. It uses schemes expressed by UML class diagrams for describing the data structure of a Data Integration System (DIS). While SIRUP focuses on the integration of data on database-level, DL-Lite uses UML class diagrams that are also commonly used for modeling software architectures. While with DL-Lite the data structures can be modeled, it does not support DDD out of the box respective in combination with SMs for an MSA.

In [Ho17], the lack of support for expressing semantics in UML is described. To address this gap, an approach for the transformation of ontologies from OWL to the Eclipse Modeling Framework (EMF)[5] is presented. However, the presented approach is tailored for the domain of space engineering and limited to the Ecore metamodel of EMF.

Another approach to embed semantics in UML diagrams is OntoUML [AG13]. OntoUML is an extension of UML 2.0 and allows the integration of ontologies on a conceptual modeling level. This approach lacks support for modeling an MSA following the DDD approach. Additionally the explicit integration of fragmented classes is not supported.

[5] `http://www.eclipse.org/modeling/emf/`

6.2 Modeling Microservice Architectures with Domain-driven Design

Based on [LDN16a] that identifies design patterns for Domain-driven Design called *Domain-driven Design Patterns* (DDDP), an approach for a *Domain-Specific Language* (DSL) for expressing domain models with Java annotations is presented in [LDN16b]. This constraints the application of DDDP to annotate a UML class diagram with DDDP DSL. Additionally, DDDP lacks support for expressing semantics of distributed domain models.

7 Conclusion and Future Work

In this paper, we presented an approach that addresses the lack of support for expressing semantic knowledge while modeling MSAs with domain-driven design. Therefore, we first describe the process of designing Microservice Architectures leveraging DDD in Section 2. Next, we introduced a metamodel that supports domain-driven MSA design in combination with semantics (cf. Section 5). Therefore, we distinguished between the external context, that is used for describing domain models of external Microservices, and Bounded Context for designing the Microservice itself. We also introduced the `DomainConcept` concept for expressing semantics in an MSA that is designed using the principles of DDD. As a formally representation of these semantics we use OWL.

In future works we plan to implement generators that produce code for Microservice models based on our metamodel and evaluate our approach by applying it to several real world projects. Next to DM and SM structures, the generated code needs to contain description and verification logic for expressed semantics. We also plan to extend our approach to not only support data integration in the context of MSAs, but also to be applicable beyond the scope of an enclosed software architecture, i.e. when different systems interact with each other.

References

[AG13] Albuquerque, Antognoni; Guizzardi, Giancarlo: An ontological foundation for conceptual modeling datatypes based on semantic reference spaces. In: 7th International Conference on Research Challenges in Information Science (RCIS 2013). IEEE, pp. 1–12, 2013.

[BPD09] Breslin, John G; Passant, Alexandre; Decker, Stefan: The Social Semantic Web. Springer, 2009.

[Ca09] Calvanese, Diego; De Giacomo, Giuseppe; Lembo, Domenico; Lenzerini, Maurizio; Rosati, Riccardo: Conceptual Modeling for Data Integration. In: Conceptual Modeling: Foundations and Applications, pp. 173–197. Springer, 2009.

[Ev04] Evans, Eric: Domain-driven Design. Tackling Complexity in the Heart of Sortware. 2004.

[FB05] Fondement, Frédéric; Baar, Thomas: Making Metamodels Aware of Concrete Syntax. In: Conceptual Modeling: Foundations and Applications, pp. 190–204. Springer, 2005.

[FLM17] Francesco, Paolo; Lago, Patricia; Malavolta, Ivano: Research on Architecting Microservices: Trends, Focus, and Potential for Industrial Adoption. In: International Conference on Software Architecture (ICSA 2017). IEEE, pp. 21–30, 2017.

[Ho17] Hoppe, Tobias; Eisenmann, Harald; Viehl, Alexander; Bringmann, Oliver: Digital Space Systems Engineering through Semantic Data Models. In: International Conference on Software Architecture (ICSA 2017). IEEE, pp. 93–96, 2017.

[LDN16a] Le, Duc Minh; Dang, Duc-Hanh; Nguyen, Viet-Ha: Domain-driven design patterns: A metadata-based approach. In: International Conference on Computing & Communication Technologies, Research, Innovation, and Vision for the Future (RIVF 2016). IEEE, pp. 247–252, 2016.

[LDN16b] Le, Duc Minh; Dang, Duc-Hanh; Nguyen, Viet-Ha: Domain-driven design using meta-attributes: A DSL-based approach. In: 8th International Conference on Knowledge and Systems Engineering (KSE 2016). IEEE, pp. 67–72, 2016.

[MPSP12] Motik, Boris; Patel-Schneider, Peter; Parsia, Bijan: , OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax, 2012. https://www.w3.org/TR/owl2-syntax/.

[Na16] Nadareishvili, Irakli; Mitra, Ronnie; McLarty, Matt; Amundsen, Mike: Microservice Architecture. O'Reilly, 2016.

[Ne15] Newman, Sam: Building Microservices. O'Reilly, 2015.

[PJ16] Pahl, Claus; Jamshidi, Pooyan: Microservices: A systematic mapping study. In: Proc. of the 6th Int. Conf. on Cloud Computing and Services Science (CLOSER 2016). pp. 137–146, 2016.

[RSZ17] Rademacher, Florian; Sachweh, Sabine; Zündorf, Albert: Differences between Model-Driven Development of Service-Oriented and Microservice Architecture. In: International Conference on Software Architecture Workshops (ICSAW). pp. 38–45, 2017.

[SS09] Staab, Steffen; Studer, Rudi, eds. Handbook on Ontologies. Springer, 2009.

[SvH06] Stuckenschmidt, H.; van Harmelen, F.: Information Sharing on the Semantic Web. Advanced Information and Knowledge Processing. Springer, 2006.

[VC16] Vresk, Tomislav; Cavrak, Igor: Architecture of an interoperable IoT platform based on microservices. In: 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO 2016). IEEE, pp. 1196–1201, 2016.

[Vi15] Villamizar, Mario; Garces, Oscar; Castro, Harold; Verano, Mauricio; Salamanca, Lorena; Casallas, Rubby; Gil, Santiago: Evaluating the monolithic and the microservice architecture pattern to deploy web applications in the cloud. In: 10th Computing Colombian Conference (CCC 2015). IEEE, pp. 583–590, 2015.

[Wo16] Wolff, Eberhard: Microservices: Flexible Software Architecture. 2016.

[YSS17] Yu, Yale; Silveira, Haydn; Sundaram, Max: A microservice based reference architecture model in the context of enterprise architecture. In: IEEE Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC 2016). IEEE, pp. 1856–1860, 2017.

[ZD07] Ziegler, Patrick; Dittrich, Klaus R: Data Integration — Problems, Approaches, and Perspectives. In: Conceptual Modelling in Information Systems Engineering, pp. 39–58. Springer, 2007.