Using Ontologies with UML Class-based Modeling: The TwoUse Approach

Fernando Silva Parreiras, Steffen Staab

ISWeb — Information Systems and Semantic Web,
Institute for Computer Science, University of Koblenz-Landau
Universitaetsstrasse 1, Koblenz 56070, Germany
{parreiras, staab}@uni-koblenz.de

Abstract

UML class-based models and OWL ontologies constitute modeling approaches with different strength and weaknesses that make them appropriate for specifying *different aspects* of software systems. We propose an integrated use of both modeling approaches in a coherent framework – *TwoUse*. We present a framework involving different concrete syntaxes for developing integrated models and use an OCL-like approach for writing query operations. We illustrate TwoUse's applicability with a case study and conclude that TwoUse achieves enhancements of nonfunctional software requirements like maintainability, reusability and extensibility.

Key words: CASE tools + UML, Ontologies, languages

1. Introduction

The Unified Modeling Language (UML)[1] is a visual design notation for designing software systems. UML is a general-purpose modeling language, capable of capturing information about different views of systems like static structure and dynamic behavior.

On the other side, ontologies provide shared domain conceptualizations representing knowledge by a vocabulary and, typically, logical definitions [2, 3] to model the problem domain as well as the solution domain. The Web Ontology Language (OWL) [4] provides a class definition language for ontologies, i.e., OWL allows for the definition of classes by required and implied logical constraints on properties of their members.

UML class-based modeling and OWL comprise some constituents that are similar in many respects like classes, associations, properties, packages, types, generalization and instances [5]. Despite of the similarities, both approaches present restrictions that may be overcome by an integration.

On the one hand, a key limitation of UML class-based modeling is that it allows only static specification of specialization and generalization of classes and relationships, whereas OWL provides mechanisms to define these as dynamic.

In other words, OWL allows for recognition of generalization and specialization between classes as well as class membership of objects based on conditions imposed on properties of class definitions.

On the other hand, UML provides means to specify dynamic behavior whereas OWL does not. The Object Constraint Language (OCL) [6] complements UML by allowing the specification of query operations, derived values, constraints, pre and post conditions.

Since both approaches provide complementary benefits, contemporary software development should make use of both. The benefits of an integration are twofold. Firstly, it provides software developers with more modeling power. Secondly, it enables semantic software developers to use object-oriented concepts like inheritance, operation and polymorphism together with ontologies in a platform independent way. These considerations lead us to the following challenge: How can we develop and denote models that benefit from advantages of the two modeling paradigms?

While mappings from one modeling paradigm to the other one have been established a while ago (e.g., [5]), the task of an integrated language for UML and OWL models has not been dealt with before. The challenge of this task arises from the large number of differing properties germane to each of the two modeling paradigms (see [7] for an analysis), making the integration of UML models and OWL models difficult.

Such an integration is not only intriguing because of the heterogeneity of the two modeling approaches, but it is now a strict requirement to allow for the development of software with many thousands of ontology classes and multiple dozens of complex software modules in the realms of medical informatics [8], multimedia [9] or engineering applications [10].

TwoUse (Transforming and Weaving Ontologies and UML in Software Engineering) addresses these types of systems. It is an approach combining UML class-based models with OWL ontologies to leverage the unique and potentially complementary strengths of the two. TwoUse's building blocks are: *(i)* an integration of the MOF-based metamodels for UML and OWL, *(ii)* the specification of dynamic behavior referring to OWL reasoning (using OCL-like expressions), *(iii)* the definition of a joint profile for denoting hybrid models as well as other concrete syntaxes.

We preset TwoUse in this paper as follows: firstly, we describe the motivation illustrated by a case study (Section 2) we have conducted. Section 3 presents and explains the building blocks of TwoUse. In Section 5, we analyze TwoUse by evaluating it according to ISO 9126 non-functional software requirements (Section 5.1) and by describing its limitations. Section 6 presents the related work. We point to further applications of TwoUse in Section 7.

2. Case Study

We use our case study in the context of semantic multimedia tool as practical running example in this paper. This is just one of the multiple uses of integrating ontologies and software modeling techniques (see Sect. 5 for further ones).

The K-Space Annotation Tool (KAT)[1] is a framework for semi-automatic and efficient annotation of multimedia content that provides (i) a plug-in Infrastructure (analysis plug-ins and visual plug-ins) and a formal model based on the Core Ontology for Multimedia (COMM) [11].

Analysis plug-ins provide functionalities to analyze content, e.g., to semi-automatically annotate multimedia data like images or videos, or to detect structure within multimedia data. However, as the number of available plug-ins increases, it becomes difficult for KAT end-users to choose appropriate plug-ins.

For example, K-Space project partners[2] provide machine learning based classifies, e.g. Support Vector Machines (SVM), for pattern recognition. There are different recognizers (object recognizers, face detectors and speaker identifiers) for different themes (sport, politics and art) for different types of multimedia data (image, audio and video) and for different formats (JPEG, GIF and MPEG). Moreover, the list of recognizers is continuously extended and, like the list of multimedia formats, it is not closed but, by sheer principle, it needs to be open.

Therefore, the objective of this use case is to provide KAT end-users with the functionality of automatically selecting and running the most appropriate plug-in(s). Such improvement enhances user satisfaction, since it prevents KAT end-users from employing unsuitable recognizers over multimedia data.

In the following, we consider three recognizers: highlight recognizer, jubilation recognizer and goal shots detector. A highlight recognizer works on detecting sets of frames in videos with high changing rates, e.g., intervals where the camera view changes frequently in a soccer game. A jubilation recognizer analyzes the video and audio, searching for shouts of jubilation. Finally, a goal shots detector works on matching shouts of jubilation with changes in camera view to characterize goal shots.

2.1. UML-based software development

An extensible approach to model recognizer variation may be applied, namely an adaptation of the Strategy Pattern [12]. The Strategy Pattern allows for encapsulating different recognizers uniformly, as depicted in Fig. 1.

Figure 1 depicts the KAT domain in the UML class diagram. It is a complex domain since KAT uses the COMM ontology that comprises many occurrences of ontology design patterns, e.g., *semantic annotation* used in the running example.

Users select KAT algorithms for SVM recognition and consequently the class controller invokes the method run() in the class kat_algorithm (Fig. 1). The method run() invokes the method getRecognizers(), which uses reflection to get a collection (rNames()) of the recognizers (_r) applicable to a given multimedia content (multimedia_data). Then, the method recognize() of each recognizer is invoked, which adds further annotations to multimedia data to refine the description (not shown in this paper for the sake of simplification).

[1] http://isweb.uni-koblenz.de/Research/kat

[2] http://www.k-space.eu/

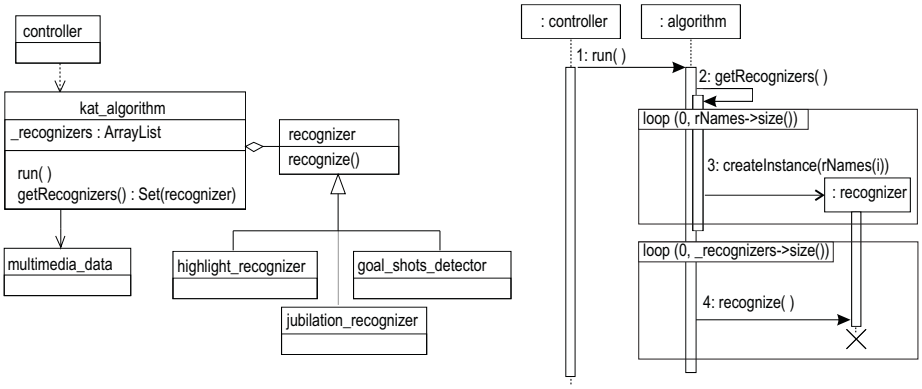

Figure 1: UML class diagram and Sequence Diagram of KAT Algorithms.

Nevertheless, applying the strategy design pattern opens the problem of strategy selection. To solve it, one needs to model how to select the most appropriate recognizer(s) to a given item of multimedia content. Listing 1 illustrates a solution using OCL. It shows the description of the query operation rNames() in OCL. This operation is used in the guard expression of the loop combined fragment in the sequence diagram (Fig. 1).

The operation rNames() collects the classes of recognizers to be created. The OCL expression Set(OclType) (Line 4) is used here as a reflection mechanism to get a list of the classes to be created. It is required to iterate through the instances of kat_algorithm (Line 4) and test if it satisfies some requirements of a given recognizer. If it does, the recognizer is added into a collection of recognizers to be created (Line 17).

Listing 1: OCL Expressions for the UML Sequence Diagram of Fig. 1

```
1 context kat_algorithm
  def rNames() : Set(OclType)
   = kat_algorithm.allInstances()
      ->iterate ( _i : kat_algorithm;
5                 _r : Set(OclType) = Set{} |
          if
           _i.annotated_data_role->exists ( adr |
             adr.video_data->exists ( v |
               v.oclIsTypeOf (soccer_video) and
10             v.semantic_annotation->exists (sa |
                 sa.kat_thing->exists ( g |
                   g.oclIsTypeOf (highlight) ) ) and
               v.semantic_annotation->exists (sa |
                 sa.kat_thing->exists ( j |
15                 j.oclIsTypeOf (jubilation) ) )
               ) )
          then
            _r->including(goal_shots_detector)
          else if
20          _i.annotated_data_role->exists ( adr |
              adr.video_data->exists ( v |
```

```
                v.oclIsTypeOf(video_data) ) )
            then
              _r->including(highlight_recognizer)->union(
              _r->including(jubilation_recognizer))
            else
              _r
            endif
          endif)->asSet()
```

In fact, the OCL expressions in Listing 1 contain class descriptions in some sense. For example, the classes highlight_recognizer and jubilation_recognizer need a kat_algorithm with some annotated_data_role with some video_data (Lines 19-24). The description of a goal_shots_detector is much more complicated (Lines 7-15), since it needs a soccer_video, that is a subclass of video_data, with some semantic_annotation with some highlight, and with some semantic_annotation with some jubilation.

Indeed, the UML/OCL approach has some limitations:

- It restricts the information that can be known about objects to object types, i.e., known information about objects is limited by information in object types (or in object states when using OCL).
- Class descriptions, e.g. goal_shots_detector (Lines 7-16), are embedded within conditional statements that are hard to maintain and reuse. In scenarios with thousands of classes, it becomes more difficult to find those descriptions, achievable only by text search.
- OCL lacks of support to transitive closure of relations [13, 14]. It makes expressions including properties like part-of more complex.

2.2. OWL-based software development

2.2.1. OWL Modeling.

Instead of hard-coding class descriptions using OCL expressions, a more expressive and extensible manner of modeling data would provide flexible ways to describe classes and, based on such descriptions, it would enable type inference.

Therefore, one requires a logical class definition language that is more expressive than UML class-based modeling. Among ontology languages, the Web Ontology Language (OWL) [4][3] is the most prominent for Semantic Web applications. Indeed, OWL provides various means for describing classes, which may also be nested into each other such that explicit typing is not compulsory. One may denote a class by a class identifier, an exhaustive enumeration of individuals, property restrictions, an intersection of class descriptions, a union of class descriptions, or the complement of a class description.

Notations for modeling OWL ontologies have been developed, resulting in textual notations [15, 22] as well as in using UML as visual notation [16, 17, 5].

[3] In this paper, we refer to the family of decidable OWL Profiles as simply OWL.

$$jubilation, highlight \sqsubseteq kat\_thing \quad (1)$$

$$soccer\_video \sqsubseteq video\_data \quad (2)$$

$$highlight\_annotation \equiv semantic\_annotation \sqcap \exists setting\_for.highlight \quad (3)$$

$$highlight\_video \equiv video\_data \sqcap \exists setting.highlight\_annotation \quad (4)$$

$$jubilation\_video \equiv video\_data \sqcap \exists setting.jubilation\_annotation \quad (5)$$

$$soccer\_jub\_hl\_video \equiv soccer\_video \sqcap highlight\_video \sqcap jubilation\_video \quad (6)$$

$$highlight\_recognizer \sqsubseteq kat\_algorithm \quad (7)$$

$$highlight\_recognizer \equiv kat\_algorithm \sqcap \exists defines(annotated\_data\_role \sqcap \exists played\_by.video\_data) \quad (8)$$

$$goal\_shots\_detector \sqsubseteq kat\_algorithm \quad (9)$$

$$goal\_shots\_detector \equiv kat\_algorithm \sqcap \exists defines(annotated\_data\_role \sqcap \exists played\_by.soccer\_jub\_hl\_video) \quad (10)$$

Table 1: KAT domain specified with Description Logics syntax.

For the sake of illustration, we use Description Logics syntax to specify the KAT domain as follows. KAT uses the COMM ontology [11] as conceptually sound model of MPEG-7 and as common but extensible denominator for different plug-ins exchanging data.

For example, the classes jubilation and highlight are subclasses of kat_thing(1). A soccer_video is a subclass of video_data(2). A highlight_annotation is a semantic_annotation that setting_for some highlight (3). A highlight_video is equivalent to a video_data that setting some highlight_annotation(4). A jubilation_video is similarly described (5). A highlight_recognizer is subclass of a kat_algorithm and is equivalent to a kat_algorithm that defines some annotated_data_role that is played_by some video_data(7-8).

OWL is compositional, i.e., OWL allows for easily reusing class descriptions to create new ones. Let us consider the class soccer_jub_hl_video(6). It is equivalent to an intersection of soccer_video, highlight_video and jubilation_video, i.e., a soccer video with highlight and jubilation. Thus, it becomes much easier to describe the class goal_shots_detector(9-10), which is subclass of a kat_algorithm and is equivalent to a kat_algorithm that defines some annotated_data_role that is played_by some soccer_jub_hl_video. Moreover, OWL allows us to define properties as transitive, simplifying query expressions. The reader may compare these reusable class definitions (even if the new language may need a bit of training) against the involved and useable implicit definition of distinctions provided in Listing 1 (Lines 6-25).

2.2.2. OWL Reasoning.

OWL ontologies can be operated on by reasoners providing services like consistency checking, concept satisfiability, instance classification and concept

classification. The reasoner performs model checking such that entailments of the Tarski-style model theory of OWL are fulfilled. For instance, we may verify whether it is possible to apply goal_shots_detector to images (consistency checking)(the answer is 'no' if goal_shots_detector is disjoint from image recognizers), or whether a given instance is a soccer_jub_hl_video(instance classification). We may ask a reasoner to classify the concepts of the ontology and find that highlight_video and jubilation_video are both superclasses of soccer_jub_hl_video (concept classification).

More specifically, given that we know an object to be an instance of highlight_video, we can infer that this object has the property setting and the value of setting is an individual of highlight_annotation. Conversely, if we have an object of video_data, which has the property setting and the value of setting associated with such individual is a highlight_annotation, we can infer that the prior individual is an instance of highlight_video. This example illustrates how one may define OWL classes like highlight_video by conditions that may be necessary as well as sufficient.

Open vs. Closed World Assumption. While the underlying semantics of UML-based class modeling adopts the closed world assumption, OWL adopts open world assumption by default. However, research in the field of combining description logics and logic programming [18] provides solutions to support OWL reasoning with closed world assumption. Different strategies have been explored like adopting an epistemic operator [19], already supported by the tableau-based OWL reasoner Pellet [20, 21]. Thus, it allows us to avoid the semantic clash in merging the two languages that might increase complexity.

To sum up, OWL provides important features complementary to UML and OCL that would improve software modeling: it allows different ways of describing classes; it handles these descriptions as first-class entities; it provides additional constructs like transitive closure for properties; and it enables dynamic classification of objects based upon class descriptions.

The need for an integration emerges since OWL is a purely declarative and logical language and not suitable to describe, e.g., dynamic aspects of software systems such as states or message passing. Thus, to benefit from inference, one must decide at which state or given which trigger one should call the reasoner. In the next section, we address this issue among others, proposing ways of integrating both paradigms using our original TwoUse approach.

3. The TwoUse Approach

We build the TwoUse approach based on four core ideas:

1. As abstract syntax, it provides an integrated *MOF based metamodel* as a common backbone for UML (including OCL) and OWL modeling.
2. As concrete syntaxes, it uses pure UML, an *UML profile* supporting standard UML2 extension mechanisms, a weaving metamodel for integrating existing UML and OWL models, and a textual concrete syntax to write UML-based class and OWL descriptions.

3. It provides a canonical *set of transformation rules* in order to deal with integration at the semantic level.
4. It provides a novel OCL-like language to write queries and constraints over OWL ontologies, OCL-DL.

To give an idea of the integration, let us consider our case study. Instead of defining the query operation rNames using UML/OCL expressions, we use the expressiveness of the OWL language together with OCL-DL. Querying an *OWL reasoning service*, an OCL-DL expression may just ask which OWL subclasses of kat_algorithm describe a given instance, enabling dynamic classification. Such expression will then be specified by:

```
1 context kat_algorithm
  def rNames(): Set(OwlType)
    self.owlSubClassesOf(kat_algorithm)->asSet()
```

As specified above, to identify which subclasses are applicable, we use an OCL-DL operation called owlSubClassesOf, which queries a reasoner to return a set of consistent named subclasses of kat_algorithm according to the OWL ontology. We explain this and other core operations in Sect. 3.4.

The advantage of this integrated formulation of rNames lies in separating two sources of specification complexity. First, the classification of complex classes remains in an OWL model. The classification reuses the COMM model and it is easily re-useable for specifying other operations; it may be maintained using graphical notations; and it is decidable, yet rigorous reasoning model (See Fig. 3). Second, the specification of the execution logic remains in the UML specification (Sequence Diagram in Fig. 1).

3.1. TwoUse Conceptual Architecture

Figure 2 presents a model-driven view of the TwoUse approach. TwoUse uses UML profiled class diagrams as concrete syntax for designing combined models. The UML class diagrams profiled for TwoUse are input for model transformations that generate TwoUse models conforming to the TwoUse metamodel. The TwoUse metamodel provides the abstract syntax for the TwoUse approach, since we have explored different concrete syntaxes (see Sect. 4). Further model transformations take TwoUse models and generate the OWL ontology and Java code.

3.2. Concrete Syntax

The TwoUse approach provides developers with an UML profile as concrete syntax for simultaneous design of UML models and OWL ontologies (UML profile for TwoUse), exploring then the full expressiveness of OWL ($\mathcal{SROIQ}(\mathcal{D})$) and allowing usage of existing UML2 tools. We reuse the UML profile for OWL proposed by OMG[5] and introduce stereotypes to label integrated classes.

We call *hybrid diagram* a UML class diagram with elements stereotyped by both UML profiles. The hybrid diagram enables different modeling views: (1) the UML view, (2) the OWL view with logical class definitions and (3)

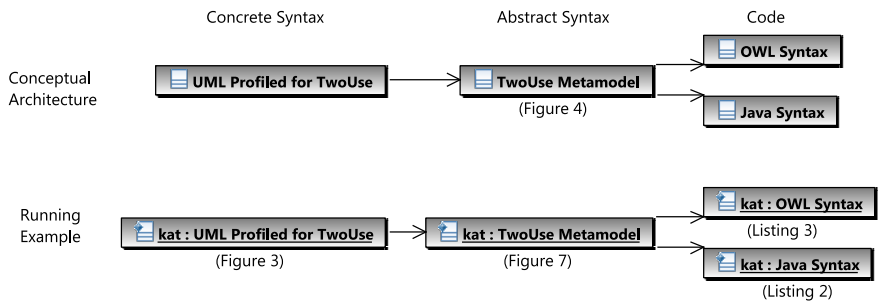

Figure 2: TwoUse Conceptual Architecture.

the TwoUse view, which integrates UML classes and OWL classes and defines OCL-DL expressions that use reasoning services.

Figure 3 shows a snippet of the UML class diagram profiled for TwoUse of our running example. In this snippet, the OWL view consists of five classes. The UML View comprises the seven classes depicted in Fig. 1 and the TwoUse view contains six classes and an OCL-DL query expression.

The TwoUse classes bridge OWL elements with OCL-DL expressions. The OCL-DL expressions are specified with the stereotype <<ocldlExpression>>. This stereotype has the property referredOwlClass [*] and the values are the classes referred by the OCL-DL expressions. Model transformations use such references to transform profiled classes into TwoUse classes.

3.3. Metamodel

The TwoUse metamodel (Fig. 4) provides the abstract syntax for describing classes with OWL and to specify OCL-like expressions that use predefined operations for reasoning over OWL models in a platform independent way. The abstract syntax provides abstraction over different concrete syntaxes used in TwoUse.

TwoUse uses the OWL2 metamodel [22] and uses package Classes::Kernel of the UML2 [1] metamodel and package BasicOCL of the OCL [6] metamodel. The OWL metamodel allows for describing classes with OWL expressiveness whereas the UML2 package Classes::Kernel allows for specifying behavioral and structural features of classes. The OCL package BasicOCL is adapted to specify OCL-DL expressions.

We integrate the OWL metamodel and the UML/OCL metamodels by composition. We apply the Object Adapter Pattern [12] to *adapt* the OWL metaclasses listed in Table 2 (Annex) to the corresponding UML Metaclasses (see [5, 7] for common features between UML and OWL). The Object Adapter Pattern allows us to compose OWL metaclass objects within *Adapters*, called TwoUse metaclasses.

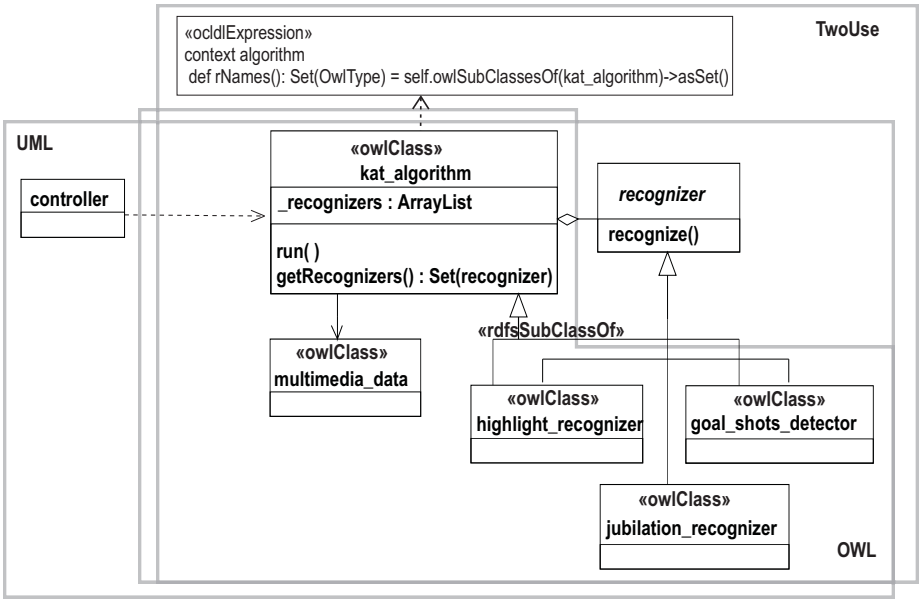

Figure 3: UML class diagram profiled with UML Profile for OWL and TwoUse Profile.

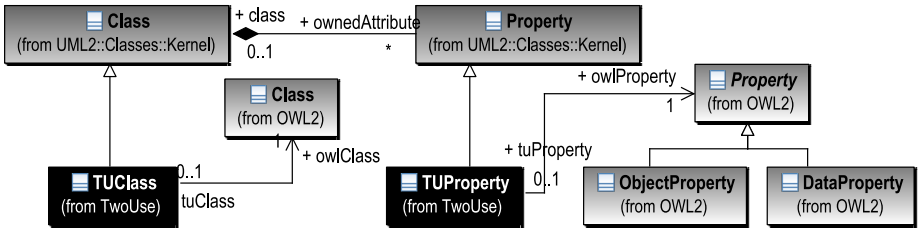

Figure 4: Excerpt of the TwoUse Metamodel (M2).

In Fig. 4, we highlight two M2 metaclasses[4] from the TwoUse metamodel: The metaclass TUClass and the metaclass TUProperty. The metaclass TUClass adapts OWL2::Class to an UML2::Class. The metaclass TUProperty is also an Adapter composing both UML2::Property and OWL2::Property, which is specialized by ObjectProperty and DataProperty. The complete metamodel is available on the TwoUse website [5].

[4]We use the terms M3, M2, M1 and M0 to refer to the corresponding layers of the OMG four-layered architecture.

[5]`http://isweb.uni-koblenz.de/Projects/twouse`

3.4. OCL-DL

OCL-DL allows for specifying expressions that rely on inferences over OWL class descriptions. It is a seamless extension of OCL towards OWL. OCL-DL uses the OCL metamodel and extends it by adding new types to the OCL types, e.g., TUClass; and a M1 model library with predefined classes and operations that enables usage of reasoning services.

OCL-DL expressions operate on snapshots of the system (M0 instances). A snapshot is the static configuration of a system at a given point in time [1], consisting of objects, values and links. In Annex A, we present an object diagram representing a possible snapshot for the running example (Fig. 6) and examples of evaluations of OCL-DL expressions (Table 3).

3.4.1. Model Library

TwoUse includes a M1 model library of predefined operations to be used in OCL-DL expressions. For example, one may replace Lines 9-15 of Listing 1 with the OCL-DL expression video.owlIsInstanceOf(soccer_jub_hl_video), which uses a reasoner to evaluate to true if the context object satisfies the sufficient conditions to be a member of class soccer_jub_hl_video. The following are the core operations of OCL-DL:

- owlIsInstanceOf(typespec: OwlType): Boolean. Evaluates to true if the context object satisfies all logical requirements of the OWL class description typespec, i.e., if $object \in typespec$.
- owlAllClasses(): Set(OwlType). It returns a set of named classes that consistently classify the context object, i.e., $\{C_1, ..., C_n\}$ where $foreach\, C, object \in C$.
- owlSubClassesOf(typespec: OwlType): Set(OwlType). It is a syntax sugar to return all named subclasses of typespec that consistently classify the context object, i.e., $\{C_1, ..., C_n\}$ where $foreach\, C, object \in C, C \sqsubseteq typespec$ and $C \neq typespec$.
- owlAllInstances(): Set(T). This is an introspective operation which returns all M0 instances that satisfy the logical requirements of the OWL class descriptions of the context object, where T is the type of the object, i.e., $\{a_1, ..., a_n\}$ where $object \in T, a \in T$, and $C \neq Thing$.
- owlMostSpecNamedClass(): OwlType. Returns the most specific named class that describes the context object, i.e., the intersection of owlSubClassesOf(typespec: OwlType). If the intersection corresponds to Nothing or to more than one most specific named class, it returns OwlInvalid.

The operations above are defined on the M1 class OwlAny of the OCL-DL library. Analogously to the OCL class OclAny, the OCL-DL class OwlAny is an instance of the OCL metaclass AnyType acting as a supertype for all TwoUse classes. Thus, every M1 TUClass inherits these operations, i.e., the operations

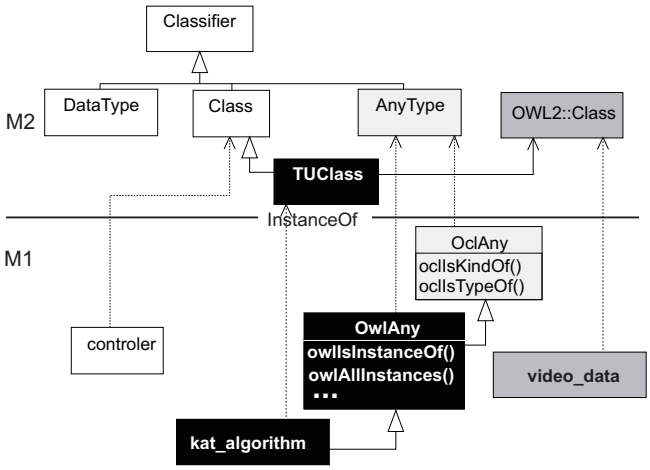

Figure 5: OCL-DL Library Extension and sample model classes.

are available on each TwoUse object in all OCL-DL expressions. Similar to OclType, OwlType is an enumeration of all basic types and OWL Classes.

Figure 5 describes OCL-DL in the context of four metamodels: OCL, UML2, OWL2 and TwoUse. At level M2, white boxes represent metaclasses used from UML2 metamodel and light grey boxes represent metaclasses from OCL metamodel. The back box represents the TwoUse metaclass TUClass from TwoUse metamodel. At level M1, we show the class kat_algorithm which is a M1 instance of the metaclass TUClass and, because of that, is a subtype of the OCL-DL class OwlAny. kat_algorithm is indirectly a M1 instance of the UML metaclass Class too and so is indirectly a subtype of the OCL Class OclAny.

Since OWL classes do not support operations, we use the TwoUse Class to build the bridge. A M1 instance of the metaclass TUClass links to a M1 instance of the metaclass OWLClass. Thus, we are able to specify OCL-DL expressions with reasoning features.

3.5. Code Generation

Generation of Java code from UML/OCL models have been explored for many years and is a straightforward task. Approaches like [23, 24] or Dresden OCL Toolkit present solutions for generation of java code from UML/OCL models.

What may seem intriguing is the mapping between Java objects and OWL. Nevertheless, these mappings have been covered many times in the literature [25, 32, 33] and the most prominent solution is Som(m)er[6], a library for mapping Java Objects to RDF graphs.

[6] `https://sommer.dev.java.net/sommer/`

In the next, we highlight how these mappings take place and how to implement OCL-DL expressions. Listing 2 depicts a simplified version of generated Java code for the class kat_algorithm. For mapping Java objects to OWL classes, Som(m)mer uses Java annotations (Lines 1,5). The Java class Ka_algorithm (Line 2) is annotated with the URI of the corresponding OWL class (Line 1). Java instances of the class Kat_algorithm at runtime correspond to individuals in the ontology. The annotation rdf.sameAs on property id (Lines 5-6) allows for identifying these individuals.

The method owlSubClassesOf implements the OCL-DL expression owlSubClassesOf up Line 17. A SPARQL query is formulated and evaluated based on the id of the current object and the name of class given as parameter to the method owlSubClassesOf (Line 18-20). URIs are then mapped into Java classes (Line 21). The remaining methods realize the operations declared in the UML class diagram profiled for TwoUse (Fig. 3).

Listing 2: Snippet of Java Code implementing TwoUse

```
1  @rdf("<http://kat-comm.owl#ka_algorithm>")
   public class Kat_algorithm {
     SesameMemorySailInit init = new SesameMemorySailInit("rdf/");
     SesameMapper mppr =
         (SesameMapper)MapperManager.getMapperForGraph(init,
         "http://www.test.de/graph1");
5    @rdf(rdf.sameAs)
     private URI id;
     private ArrayList<Recognizer> _recognizers;
     public ArrayList<Class> rNames(){
        return this.owlSubClassesOf("kat_algorithm");}
10   public void run(){
        for (Recognizer r: getRecognizers()){
           r.recognize();}}
     public ArrayList<Recognizer> getRecognizers(){
        for (Class c: rNames()) {
15         _recognizers.add((Recognizer) c.newInstance());}
        return _recognizers;}
     public ArrayList<Class> owlSubClassesOf(String typespec){
        String query = "SELECT DISTINCT ?c WHERE{ "+ id +" rdf:type
            ?c . ?c rdfs:subClassOf :" + typespec + " FILTER (?c !=
            :"+ typespec +")}";
//      .. query execution here
20      for (URI u: ul) {
           List<Class> a = mppr.getClassesOf(u.toString());
           cls.add(a.get(0));}
        return cls;}}
```

3.6. Implementation

We have been implemented the TwoUse approach in the Eclipse Platform using the Eclipse Modeling Framework [26] and is available for download on the project website.

To realize the concrete syntaxes, we have adopted implementations of UML2 and OCL from the Model Development Tools (MDT) project [24]. As UML editor, we use the open source Papyrus UML. To handle model transformations

and serializations, we use tools from the Eclipse/GMT project (AMW, TCS, KM3 and ATL) [27].

In Java code, we use the free open-source reasoner Pellet [20] for reasoning with OWL. Som(m)er is responsible for realizing the integration between OWL and Java, as shown above.

4. Alternative Concrete Syntaxes

Based on feedback from potential TwoUse users, we have explored additional concrete syntaxes with increasing expressiveness, presented next. Additionally to the UML Profile for TwoUse presented in Sect 3.2, one may use the pure UML class diagram notation to model OWL ontologies with OCL-DL expressions at class operations; or use a weaving metamodel to weave existing UML models and OWL ontologies and add OCL-DL expressions; or, finally, use a textual syntax to design class-based models with OWL descriptions. We provide model transformations from the different concrete syntaxes into TwoUse models.

4.1. Pure UML Class Diagrams

To quickly start UML2 users developing semantic web applications, pure UML class diagrams may be used. Developers who do not need the full expressiveness of OWL can use this approach without having to handle the OWL syntax.

Model transformations transform the UML class diagram into a TwoUse model to support OCL-DL expressions over the OWL translation of the UML class diagram. In this case, developers attach OCL-DL expressions to BODY constraints (Opaque Expressions) on class operations. Each UML class will be a TUClass, i.e., it is an UML class with link to an OWL class. For transforming UML class diagrams into ontologies, we follow the rules defined in [5] (Chapter 16) [7].

4.2. Weaving Model

When existing UML models and OWL ontologies are available, we provide developers with a more productive approach than modeling UML class diagrams. In this case, they can use a Model Weaver [27] to create mappings between existing UML classes and OWL classes.

Developers annotate the UML classes to be woven with OWL classes. The weaving model capture such annotations and allows for writing OCL-DL expressions. The weaving model together with both UML and OWL models serves as input to transformation into TwoUse Models. The weaving metamodel is also available on the TwoUse website.

[7] In this case, the expressiveness of the generated OWL ontology is limited to the description logics $\mathcal{ALCOIQ}(\mathcal{D})$, since $\mathcal{DLR}_{ifd}$ is not supported by current state-of-the-art DL-based reasoning systems [28].

4.3. Textual Notation

As alternative to graphical languages, we defined a textual notation for specifying UML class-based models together with OWL. This approach is useful for experienced developers who work more productively with textual languages than visual languages.

In the following, we illustrate the textual notation with our running example. A query operation rNames() is defined for the class kat_algorithm. Again, each class is a TUClass. In this case, the textual notation allows for exploring the full expressiveness of OWL.

```
class kat_algorithm extends core:algorithm{
  query rNames(): Set(OwlType)
     = self.owlSubClasses(kat_algorithm)->asSet();
}
...
class highlight_annotation equivalentTo (core:semantic_annotation
    restrictionOn dsn:setting_for some highlight) {}
class highlight_video equivalentTo (core:video_data restrictionOn
    dsn:setting some highlight_annotation){}
class jubilation_video equivalentTo (core:video_data
    restrictionOn dsn:setting some jubilation_annotation){}
class soccer_jub_hl_video equivalentTo ( ObjectIntersectionOf
    (soccer_video highlight_video jubilation_video)){}
class highlight_recognizer extends kat_algorithm, restrictionOn
    dns:defines some (core:annotated_data_role restrictionOn
    played_by some core:video_data){}
class jubilation_recognizer extends kat_algorithm, restrictionOn
    dns:defines some (core:annotated_data_role restrictionOn
    dns:played_by some core:video_data){}
class goal_shots_detector extends kat_algorithm, restrictionOn
    dns:defines some (core:annotated_data_role restrictionOn
    dns:played_by some soccer_jub_hl_video){}
```

The textual notation uses constructs familiar to programmers and enables developers to write class descriptions in a human readable way instead of XMI. We have been used the textual notation for writing integrated models for a telecommunication company, where the usage of OWL for variability management plays an important role. [8].

5. Discussion

5.1. TwoUse Analysis

Based on the case study, we analyze how TwoUse features reflect development-oriented non-functional requirements according to a quality model covering the following quality factors: maintainability, efficiency (ISO 9126 [29]), reusability and extensibility [30]. The decision of using UML with OWL does not affect other ISO 9126 quality factors.

[8] For more information, visit http://isweb.uni-koblenz.de/Research/OBDF

Maintainability. We analyze Maintainability with regard to analyzability, changeability and testability as follows.

Analyzability. In case of failure in the software, the developers might firstly check the consistency of the domain and then use axiom explanation to track down failure, which helps to improve failure analysis efficiency. These services are already available in ontology tools.

Changeability. The knowledge encoded in OWL evolves independently of the execution logic, i.e., developers maintain class descriptions in the ontology and not in the software. Since the software does not need recompilation and redistribution, the work time spent to change decreases.

Testability. An OWL reasoner verifies queries an class definitions declared in unit tests[9] to test ontology axioms, enabling test suites to be more declarative.

Reusability. Extending the COMM core ontology allows developers to reuse available knowledge about multimedia content, semantic annotation and algorithm. Furthermore, developers can reuse the knowledge represented in OWL independently of platform or programming language.

Moreover, developers may use class descriptions to semantically query the domain. Semantic query plays an important role in large domains like KAT (approx. 750 classes). For example, developers may want to reuse algorithm descriptions applicable to videos. By executing the query

$$\sqcap \exists defines(annotated\_data\_role \sqcap \exists played\_by.video)? \tag{11}$$

developers would see that the classes highlight_recognizer, jubilation_recognizer and goal_shots_detector are candidate to reuse. Such a semantic query is not possible with UML/OCL.

Extensibility. When the application requires it, developers can be more specific by extending existing concepts and adding statements. For example, once developers identify that an algorithm works better with certain types of videos, they can easily extend the algorithm description.

5.2. Application Scenarios

We have applied TwoUse not only in ontology-based systems but also to development of non-logical systems as follows:

Improving General Purpose Software Design Patterns. We have explored how to improve the Strategy Pattern and Abstract Factory with ontologies in [31] and we are able to provide improvements to the Visitor Pattern and to the Bridge Pattern as well.

[9] `http://www.co-ode.org/downloads/owlunittest/`

Metamodeling. We have analyzed the OCL constraints specified in the UML2 Specification [1] and identified 6 out of 90 complex constraints that could benefit from TwoUse. Moreover, where property transitivity is required, e.g., in specifying constructs like Activity, State, StateMachine and Transition, TwoUse allows for defining additional operations that are not expressible in OCL.

Automatically generation of ontology APIs. The task of programming ontology APIs is not trivial, since ontologies usually rely on ontology design patterns (e.g. semantic annotation in the running example) that are difficult to map onto Java objects. TwoUse allows for specifying these mappings as well as queries using OCL-DL[10].

Ontology based domain specific languages. Domain specific languages (DSL) may apply dynamic classification to recommend domain concepts to novice DSL users (who may not be aware of suitable domain classes). TwoUse enables DSL users to execute OCL-DL queries that, based on OWL classes, dynamically classify content[11].

Using ontologies with variability management at runtime. In software product line engineering, ontologies can describe variants and variant constraints and OCL-DL expressions specify the decision of which variant to employ at runtime. We currently investigate this and other application scenarios under the EU project MOST[12].

5.3. Limitations

Calls from OCL to the OWL reasoner (OCL-DL) may return OWL classes that are not part of the TwoUse model. This implies a dynamic diffusion of OWL classes into the UML model, which must either be accommodated dynamically or which may need to raise an exception (in our use case, we pursue the latter).

By weaving UML and OWL ontologies, TwoUse requires sufficient understanding of developers about topics like property restriction and satisfiability. There is a trade-off between a concise and clear definition of syntax that is unknown to many people as in Table 1 vs. an involved syntax that people know. From past experiences, we conclude that, in the long run, the higher level expressivity will win and developers are willing to learn a more expressive approach.

Indeed, we have provided various syntaxes that might better or less well fit the needs of different software developers but in some cases, it does not prevent them from understanding the semantics of OWL constructs. This shortcoming is minimized in case of ontology-based applications like KAT, since software developers are already familiar with OWL.

[10] `http://isweb.uni-koblenz.de/Research/agogo`
[11] `http://isweb.uni-koblenz.de/Research/OBDF`
[12] `http://www.most-project.eu`

6. Related Work

Several MOF based metamodels for OWL, UML profiles for OWL and transformations from UML into OWL are available in literature [16, 17, 5], some of them with new adornments. These profiles and transformations are designed exclusively to model OWL ontologies with UML and they do not provide support for integrating OWL in UML modeling.

Several strategies to integrate programming code and ontologies are available as well [25, 32, 33, 34]. None of them, however, operates on the platform independent level. We propose an integration between UML and OWL regardless of programming language.

Development of Semantic Web Services uses ontologies for domain modeling and specification of behavioral properties to analyze contracts [35, 36]. However, we use OCL-like expressions to specify reasoning calls. Our approach proposes modeling constructors used at design time to specify the usage of OWL ontologies at runtime.

One might think of using reflection to dynamically classify instances based on simple class descriptions. However, with reflection it is not possible to achieve the expressiveness of OWL DL. Furthermore, the usage of reflection embeds knowledge about class descriptions into Java class definitions among non-trivial constructs. TwoUse employs the expressiveness of OWL ontologies to realize such decisions in a transparent way for software developers.

7. Conclusion

This paper proposes TwoUse as an approach enabling UML modeling with semantic expressiveness of OWL DL. We propose bridges based on a metamodel, library extensions and model transformations. TwoUse achieves improvements on the maintainability, reusability and extensibility for ontology based system development, which corroborates Description Logic literature [37]. TwoUse allows developers to raise the level of abstraction of business rules until now embedded in OCL expressions. Additionally, TwoUse allows for formalizing these rules with OWL.

Future Work.. Since OCL-DL is an extension of OCL, we are investigating how to apply Twouse into other class-based modeling approaches that use OCL. Furthermore, we are working on defining a formal semantics for TwoUse, based on existing model theoretic semantics for UML/OCL [38] and OWL [4].

References

[1] OMG: Unified Modeling Language: Superstructure, version 2.1.2. Object Modeling Group. (November 2007)

[2] Gruber, T.R.: A translation approach to portable ontology specifications. Knowledge Acquisition **5**(2) (1993) 199–220

[3] Staab, S., Studer, R., eds.: Handbook on Ontologies. International Handbooks on Information Systems. Springer (2004)

[4] W3C OWL Working Group: OWL 2 Web Ontology Language Document Overview. W3C Working Draft 27 March 2009 Available at `http://www.w3.org/TR/2009/WD-owl2-overview-20090327//`.
[5] OMG: Ontology Definition Metamodel. Object Modeling Group. (September 2008)
[6] OMG: Object Constraint Language Specification, version 2.0. Object Modeling Group. (June 2005)
[7] Silva Parreiras, F., Staab, S., Winter, A.: On Marrying Ontological and Metamodeling Technical Spaces. In: ESEC/FSE'07, Cavtat, Croatia, ACM (September 2007) 439–448
[8] O'Connor, M.J., Shankar, R., Tu, S.W., Nyulas, C., Parrish, D., Musen, M.A., Das, A.K.: Using semantic web technologies for knowledge-driven querying of biomedical data. In: AIME. (2007) 267–276
[9] Staab, S., Scherp, A., Arndt, R., Troncy, R., Gregorzek, M., Saathoff, C., Schenk, S., Hardman, L.: Semantic multimedia. In: Reasoning Web, 4th International Summer School, Venice, Italy. Volume 5224 of LNCS., Springer (2008) 125–170
[10] Staab, S., Franz, T., Görlitz, O., Saathoff, C., Schenk, S., Sizov, S.: Lifecycle knowledge management: Getting the semantics across in x-media. In: Foundations of Intelligent Systems, ISMIS 2006, Bari, Italy, September 2006. Volume 4203 of LNCS., Springers (2006) 1–10
[11] Arndt, R., Troncy, R., Staab, S., Hardman, L., Vacura, M.: COMM: Designing a well-founded multimedia ontology for the web. In: Proc. of ISWC 2007, Busan, Korea. Volume 4825 of LNCS., Springer (2007) 30–43
[12] Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design patterns: elements of reusable object-oriented software. Addison-Wesley Professional (1995)
[13] Van Der Straeten, R., Mens, T., Simmonds, J., Jonckers, V.: Using description logic to maintain consistency between uml models. In: UML 2003. Volume 2863 of LNCS., Springer 326 – 340
[14] Bodeveix, J.P., Millan, T., Percebois, C., Camus, C.L., Bazex, P., Feraud, L.: Extending OCL for verifying UML models consistency. In: Workshop on Consistency Problems in UML-based Software Development. Workshop at UML 2002. (2002) 75–90
[15] Horridge, M., Drummond, N., Goodwin, J., Rector, A., Stevens, R., Wang., H.: The Manchester OWL Syntax. In: OWL: Experiences and Directions (OWLED) 2006, Athens, Georgia, USA (November 2006)
[16] Brockmans, S., Haase, P., Hitzler, P., Studer, R.: A metamodel and UML profile for rule-extended OWL DL ontologies. In: Proc. of ESWC 2006. Volume 4011 of LNCS., Springer (2006) 303–316
[17] Djurić, D., Gašević, D., Devedžić, V., Damjanovic, V.: A UML profile for OWL ontologies. In: MDAFA. Volume 3599 of LNCS., Springer (2005) 204–219
[18] Motik, B., Horrocks, I., Rosati, R., Sattler, U.: Can owl and logic programming live together happily ever after? In: In Proc. ISWC-2006. Volume 4273 of LNCS., Springer (2006) 501–514
[19] Donini, F.M., Nardi, D., Rosati, R.: Description logics of minimal knowledge and negation as failure. ACM Trans. Comput. Logic **3**(2) (2002) 177–225

[20] Parsia, B., Sirin, E.: Pellet: An OWL DL Reasoner. In: Proc. of the 2004 International Workshop on Description Logics (DL2004). Volume 104 of CEUR Workshop Proceedings. (2004)
[21] Katz, Y., Parsia, B.: Towards a Nonmonotonic Extension to OWL. In: Proceedings of the OWLED 2005, Galway, Ireland, November 11-12, 2005. Volume 188 of CEUR Workshop Proceedings. (2005)
[22] Motik, B., Patel-Schneider, P.F., Parsia, B.: OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax. W3C Working Draft 02 December 2008
[23] Warmer, J., Kleppe, A.: The Object Constraint Language: Getting Your Models Ready for MDA. Addison-Wesley (2003)
[24] The Eclipse Foundation: The Model Development Tools (MDT) project (2007) Available at `http://www.eclipse.org/modeling/mdt/`.
[25] Eberhart, A.: Automatic generation of Java/SQL based inference engines from RDF schema and RuleML. In: Proc. of the 1st International Semantic Web Conference ISWC 2002, London, UK, Springer (2002) 102–116
[26] Budinsky, F., Brodsky, S.A., Merks, E.: Eclipse Modeling Framework. Pearson Education (2003)
[27] The Eclipse Foundation: The Generative Modeling Technologies (GMT) project (2009) Available at `http://www.eclipse.org/gmt/`.
[28] Berardi, D., Calvanese, D., Giacomo, G.D.: Reasoning on UML class diagrams. Artif. Intell. **168**(1) (2005) 70–118
[29] ISO/IEC: ISO/IEC 9126. Software engineering – Product quality. (2001)
[30] Ebert, C.: Dealing with nonfunctional requirements in large software systems. Ann. Softw. Eng. **3** (1997) 367–395
[31] Silva Parreiras, F., Staab, S., Winter, A.: Improving design patterns by description logics: A use case with abstract factory and strategy. In: Modellierung 2008. Volume P-127 of LNI., GI (2008) 89–104
[32] Kalyanpur, A., Pastor, D.J., Battle, S., Padget, J.A.: Automatic mapping of OWL ontologies into java. In SEKE (2004) 98–103
[33] Knublauch, H.: Ontology-driven software development in the context of the semantic web: An example scenario with Protege/OWL. In: 1st International Workshop on the Model-Driven Semantic Web, Monterey, California, USA (2004)
[34] Puleston, C., Parsia, B., Cunningham, J., Rector, A.: Integrating object-oriented and ontological representations: A case study in java and owl. In: Prof. of ISWC '08, October 26-30, Karlsruhe, Germany, Berlin, Heidelberg, Springer-Verlag (2008) 130–145
[35] Li, Z., Han, J., Jin, Y.: Pattern-based specification and validation of web services interaction properties. In: Proc. of ICSOC 2005, Amsterdam, The Netherlands. Volume 3826 of LNCS., Springer (2005) 73–86
[36] Sriharee, N., Senivongse, T.: Discovering web services using behavioural constraints and ontology. In: Proc of 4th IFIP WG6.1 , DAIS 2003, Paris, France. Volume 2893 of LNCS., Springer (2003) 248–259
[37] McGuinness, D.L.: Configuration. In: The Description Logic Handbook. Cambridge University Press (2003) 397–414
[38] Schmitt, P.H.: A model theoretic semantics for ocl. In: In Proc. of IJCAR Workshop on Precise Modelling and Deduction for Object-oriented Software Development (PMD. (2001)

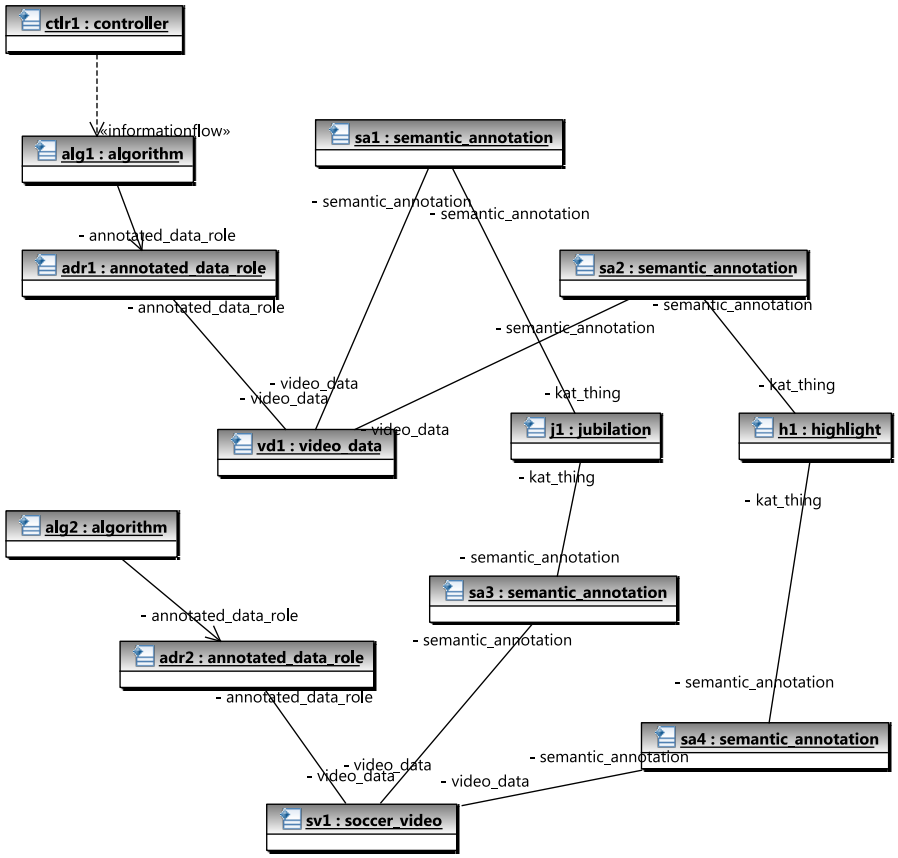

Figure 6: Snapshot of the running example (M0).

Annex A

This annex presents material supporting the running example. It comprises additional diagrams, examples of evaluating OCL-DL expressions and the list of axioms of the OWL ontology.

7.1. Snapshot

A snapshot is the static configuration of a system at a given point in time [1]. It consists of objects, values and links. To illustrate how OCL-DL expressions work, we consider the snapshot presented in Fig. 6.

7.2. TwoUse Model (Abstract Syntax)

The TwoUse abstract model is generated as output of model transformations that take models in any of the concrete syntaxes supported by TwoUse as input. Figure 8 depicts an excerpt of the abstract model for the running example.

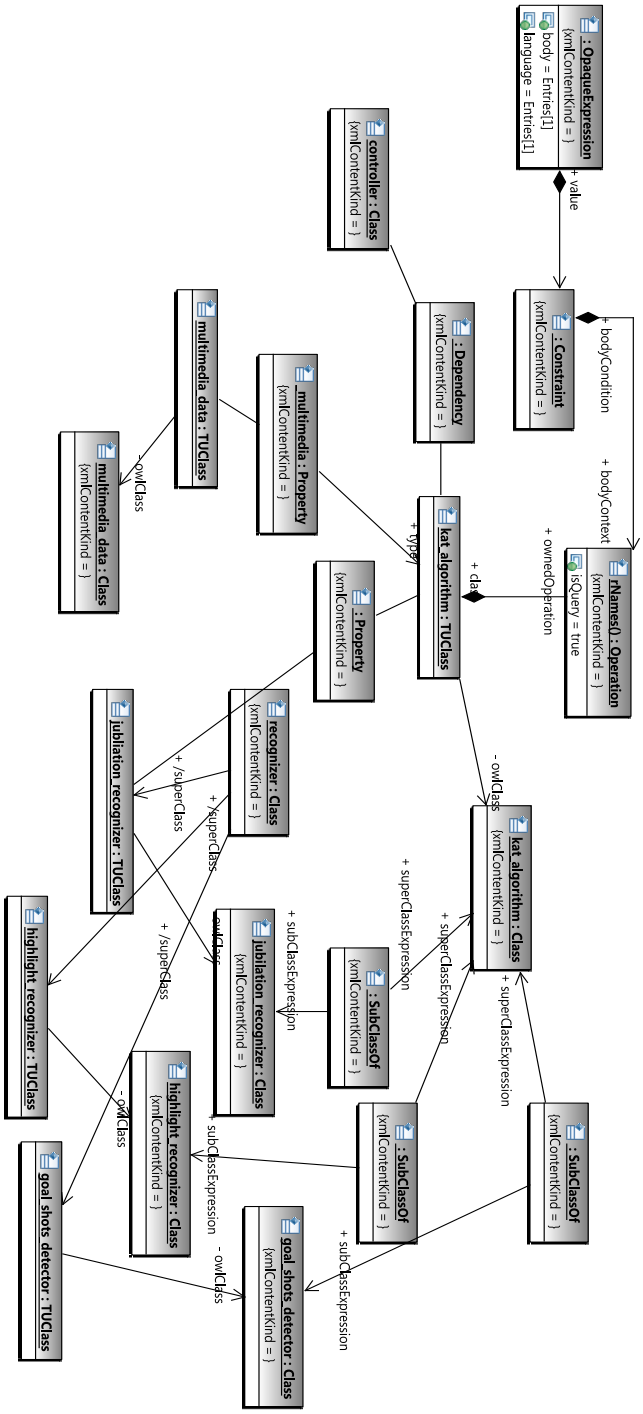

Figure 7: Excerpt of a TwoUse model (M1).

7.3. TwoUse Metamodel

The TwoUse metamodel composes the UML2 metamodel and the OWL2 metamodel by applying the strategy design pattern in classes representing common features. Table 2 lists these classes. The UML2 classes are then specialized by TwoUse classes as depicted in Fig. 8.

UML2	OWL2
Package	Ontology
Class	Class
Enumeration	ObjectOneOf, DataOneOf
Property,	ObjectProperty, DataProperty
MultiplicityElement	ObjectMinCardinality, ObjectMaxCardinality, DataMinCardinality, DataMaxCardinality
InstanceSpecification	Individual
LiteralSpecification	Literal

Table 2: Common features between UML and OWL.

7.4. Evaluation of OCL-DL expressions

Table 3 lists results of evaluating OCL-DL expressions considering the snapshot depicted in Fig. 6. We take two objects of the snapshot (alg1, alg2) and bind them to the predefined variable self. For example, for the expression self.owlIsInstanceOf(highlight_recognizer) where self is bound to alg1, the result is true.

Context object OCL-DL Expression	alg1	alg2
owlIsInstanceOf(highlight_recognizer)	true	true
owlIsInstanceOf(goal_shots_detector)	false	true
owlAllClasses()	algorithm, description, highlight_recognizer, jubilation_recognizer, method	algorithm, description, highlight_recognizer, jubilation_recognizer, goal_shots_detector, method
owlSubClassesOf(algorithm)	highlight_recognizer, jubilation_recognizer	highlight_recognizer, jubilation_recognizer, goal_shots_detector
owlAllInstances()	alg1, alg2	alg1, alg2
owlMostSpecNamedClass()	owlInvalid	goal_shots_detector

Table 3: Evaluation of OCL-DL expressions according to the running example snapshot.

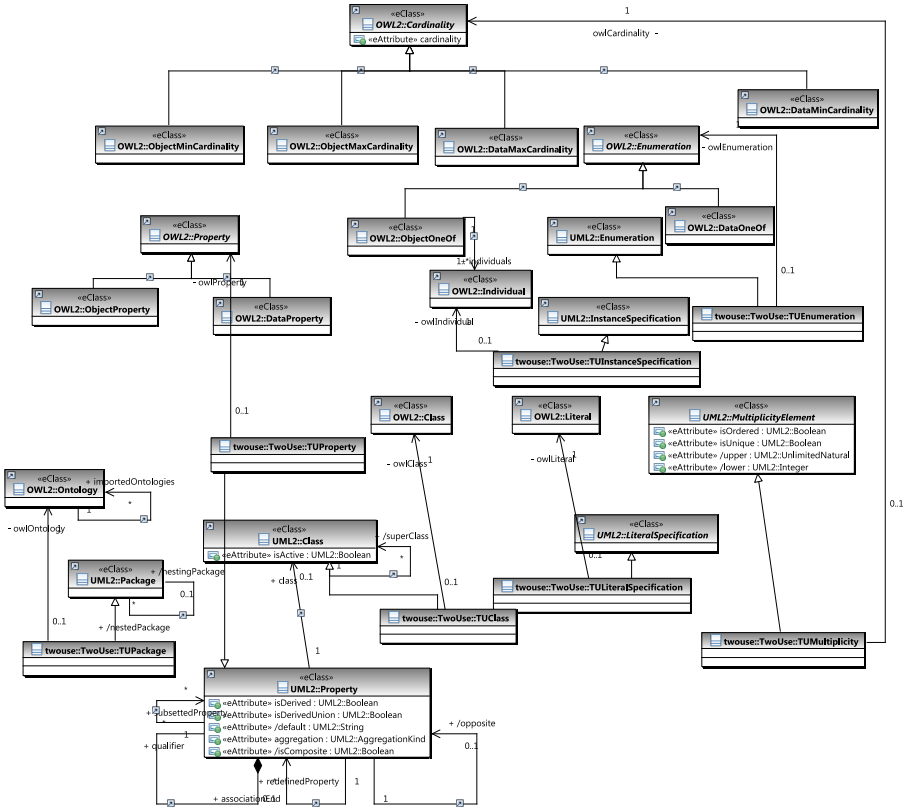

Figure 8: TwoUse Metamodel (M2).

7.5. OWL Ontology

Listing 3 presents the axioms of the KAT ontology described in the OWL2 functional syntax. The KAT ontology is available for download on TwoUse website.

Listing 3: KAT Ontology for the running example

```
1
Namespace(=<http://isweb.uni-koblenz.de/2008/4/comm-kat.owl#>)
Namespace(rdfs=<http://www.w3.org/2000/01/rdf-schema#>)
Namespace(extended-dns-very-lite=<http://comm.semanticweb.org/extended-dns-very-lite.owl#>)
5 Namespace(visual=<http://comm.semanticweb.org/visual.owl#>)
Namespace(owl2xml=<http://www.w3.org/2006/12/owl2-xml#>)
Namespace(localization=<http://comm.semanticweb.org/localization.owl#>)
Namespace(owl=<http://www.w3.org/2002/07/owl#>)
Namespace(xsd=<http://www.w3.org/2001/XMLSchema#>)
10 Namespace(comm-kat=<http://isweb.uni-koblenz.de/2008/4/comm-kat.owl#>)
Namespace(rdf=<http://www.w3.org/1999/02/22-rdf-syntax-ns#>)
Namespace(core=<http://comm.semanticweb.org/core.owl#>)
Namespace(media=<http://comm.semanticweb.org/media.owl#>)

15 Ontology(<http://isweb.uni-koblenz.de/2008/4/comm-kat.owl>

Import(<http://comm.semanticweb.org/multimedia-ontology.owl>)

// Class:
    http://isweb.uni-koblenz.de/2008/4/comm-kat.owl#jubilation
20
SubClassOf(jubilation kat-domain) DisjointClasses(accident
                goal
                grass
                highlight
25              jubilation)
// Class:
    http://isweb.uni-koblenz.de/2008/4/comm-kat.owl#grass-video-data

EquivalentClasses(grass-video-data
ObjectIntersectionOf(ObjectSomeValuesFrom(extended-dns-very-lite:setting
30 grass-semantic-annotation) core:video-data))
    SubClassOf(grass-video-data
core:video-data)
// Class: http://comm.semanticweb.org/core.owl#algorithm

// Class:
    http://isweb.uni-koblenz.de/2008/4/comm-kat.owl#goal-shots-detector
35
EquivalentClasses(goal-shots-detector
ObjectIntersectionOf(ObjectSomeValuesFrom(extended-dns-very-lite:defines
ObjectIntersectionOf(ObjectSomeValuesFrom(extended-dns-very-lite:played-by
soccer-jub-hl-video) core:annotated-data-role)) core:algorithm))
40 SubClassOf(goal-shots-detector core:algorithm)
// Class: http://comm.semanticweb.org/core.owl#annotated-data-role

// Class:
    http://isweb.uni-koblenz.de/2008/4/comm-kat.owl#highlights-video-data

45 EquivalentClasses(highlights-video-data
```

```
  ObjectIntersectionOf(ObjectSomeValuesFrom(extended-dns-very-lite:setting
  highlight-semantic-annotation) core:video-data))
  SubClassOf(highlights-video-data core:video-data)
  // Class: http://isweb.uni-koblenz.de/2008/4/comm-kat.owl#asphalt

  SubClassOf(asphalt kat-domain)
  // Class:
       http://isweb.uni-koblenz.de/2008/4/comm-kat.owl#soccer-jub-hl-video

  EquivalentClasses(soccer-jub-hl-video
  ObjectIntersectionOf(highlights-video-data
                                                          jubilation-video-data
                                                          soccer-video-data))
  SubClassOf(soccer-jub-hl-video core:video-data)
  // Class: http://comm.semanticweb.org/core.owl#semantic-annotation

  // Class: http://isweb.uni-koblenz.de/2008/4/comm-kat.owl#grass

  SubClassOf(grass kat-domain) DisjointClasses(accident
                  goal
                  grass
                  highlight
                  jubilation)
  // Class: http://isweb.uni-koblenz.de/2008/4/comm-kat.owl#accident

  SubClassOf(accident kat-domain) DisjointClasses(accident
                  goal
                  grass
                  highlight
                  jubilation)
  // Class:
       http://isweb.uni-koblenz.de/2008/4/comm-kat.owl#jubilation-recognition-algorithm

  EquivalentClasses(jubilation-recognition-algorithm
  ObjectIntersectionOf(ObjectSomeValuesFrom(extended-dns-very-lite:defines
  ObjectIntersectionOf(ObjectSomeValuesFrom(extended-dns-very-lite:played-by
  core:video-data) core:annotated-data-role)) core:algorithm))
  SubClassOf(jubilation-recognition-algorithm core:algorithm)
  // Class:
       http://comm.semanticweb.org/extended-dns-very-lite.owl#goal

  // Class:
       http://isweb.uni-koblenz.de/2008/4/comm-kat.owl#grass-image-data

  EquivalentClasses(grass-image-data
  ObjectIntersectionOf(ObjectSomeValuesFrom(extended-dns-very-lite:setting
  grass-semantic-annotation) core:image-data))
       SubClassOf(grass-image-data
  core:image-data)
  // Class: http://isweb.uni-koblenz.de/2008/4/comm-kat.owl#goal

  SubClassOf(goal kat-domain) DisjointClasses(accident
                  goal
                  grass
                  highlight
                  jubilation)
  // Class:
```

```
        http://isweb.uni-koblenz.de/2008/4/comm-kat.owl#motorsport-video-data

    EquivalentClasses(motorsport-video-data
100 ObjectIntersectionOf(ObjectSomeValuesFrom(extended-dns-very-lite:setting
    asphalt-semantic-annotation) core:video-data))
        SubClassOf(motorsport-video-data
    core:video-data)
    // Class:
        http://isweb.uni-koblenz.de/2008/4/comm-kat.owl#kat-domain

105 SubClassOf(kat-domain owl:Thing)
    // Class:
        http://isweb.uni-koblenz.de/2008/4/comm-kat.owl#grass-semantic-annotation

    EquivalentClasses(grass-semantic-annotation
    ObjectIntersectionOf(ObjectSomeValuesFrom(extended-dns-very-lite:setting-for
110 grass) core:semantic-annotation))
    // Class: http://comm.semanticweb.org/core.owl#semantic-label-role

    // Class: http://comm.semanticweb.org/core.owl#image-data

115 // Class:
        http://isweb.uni-koblenz.de/2008/4/comm-kat.owl#jubilation-video-data

    EquivalentClasses(jubilation-video-data
    ObjectIntersectionOf(ObjectSomeValuesFrom(extended-dns-very-lite:setting
    jubilation-semantic-annotation) core:video-data))
120 SubClassOf(jubilation-video-data core:video-data)
    // Class: http://isweb.uni-koblenz.de/2008/4/comm-kat.owl#highlight

    SubClassOf(highlight kat-domain) DisjointClasses(accident
                    goal
125                 grass
                    highlight
                    jubilation)
    // Class:
        http://isweb.uni-koblenz.de/2008/4/comm-kat.owl#asphalt-semantic-annotation

130 EquivalentClasses(asphalt-semantic-annotation
    ObjectIntersectionOf(ObjectSomeValuesFrom(extended-dns-very-lite:setting-for
    asphalt) core:semantic-annotation))
        SubClassOf(asphalt-semantic-annotation
    core:semantic-annotation)
    // Class: http://www.w3.org/2002/07/owl#Thing
135
    // Class:
        http://isweb.uni-koblenz.de/2008/4/comm-kat.owl#highlight-semantic-annotation

    EquivalentClasses(highlight-semantic-annotation
    ObjectIntersectionOf(ObjectSomeValuesFrom(extended-dns-very-lite:setting-for
140 highlight) core:semantic-annotation))
        SubClassOf(highlight-semantic-annotation
    core:semantic-annotation)
    // Class:
        http://isweb.uni-koblenz.de/2008/4/comm-kat.owl#goal-semantic-annotation

    EquivalentClasses(goal-semantic-annotation
```

```
ObjectIntersectionOf(ObjectSomeValuesFrom(extended-dns-very-lite:setting-for
extended-dns-very-lite:goal) core:semantic-annotation))
SubClassOf(goal-semantic-annotation core:semantic-annotation)
// Class: http://comm.semanticweb.org/core.owl#video-data

// Class:
    http://isweb.uni-koblenz.de/2008/4/comm-kat.owl#soccer-video-data

SubClassOf(soccer-video-data core:video-data)
// Class:
    http://isweb.uni-koblenz.de/2008/4/comm-kat.owl#jubilation-semantic-annotation

EquivalentClasses(jubilation-semantic-annotation
ObjectIntersectionOf(ObjectSomeValuesFrom(extended-dns-very-lite:setting-for
jubilation) core:semantic-annotation))
SubClassOf(jubilation-semantic-annotation core:semantic-annotation)
// Class:
    http://isweb.uni-koblenz.de/2008/4/comm-kat.owl#highlights-recognition-algorithm

EquivalentClasses(highlights-recognition-algorithm
ObjectIntersectionOf(ObjectSomeValuesFrom(extended-dns-very-lite:defines
ObjectIntersectionOf(ObjectSomeValuesFrom(extended-dns-very-lite:played-by
core:video-data) core:annotated-data-role)) core:algorithm))
SubClassOf(highlights-recognition-algorithm core:algorithm)
// Object property:
    http://comm.semanticweb.org/extended-dns-very-lite.owl#satisfies

// Object property:
    http://comm.semanticweb.org/extended-dns-very-lite.owl#plays

// Object property:
    http://comm.semanticweb.org/extended-dns-very-lite.owl#defines

// Object property:
    http://comm.semanticweb.org/extended-dns-very-lite.owl#played-by

// Object property:
    http://comm.semanticweb.org/extended-dns-very-lite.owl#setting

// Object property:
    http://comm.semanticweb.org/extended-dns-very-lite.owl#setting-for

// Individual: http://isweb.uni-koblenz.de/2008/4/comm-kat.owl#slr1

ClassAssertion(slr1 core:semantic-label-role)
// Individual: http://isweb.uni-koblenz.de/2008/4/comm-kat.owl#slr2

ClassAssertion(slr2 core:semantic-label-role)
// Individual: http://isweb.uni-koblenz.de/2008/4/comm-kat.owl#h1

ClassAssertion(h1 highlight)
// Individual: http://isweb.uni-koblenz.de/2008/4/comm-kat.owl#j1

ClassAssertion(j1 jubilation)
ObjectPropertyAssertion(extended-dns-very-lite:plays j1 slr1)
// Individual: http://isweb.uni-koblenz.de/2008/4/comm-kat.owl#alg2
```

```
  ClassAssertion(alg2 core:algorithm)
  ObjectPropertyAssertion(extended-dns-very-lite:defines alg2 adr2)
195 ObjectPropertyAssertion(extended-dns-very-lite:defines alg2 slr2)
  // Individual: http://isweb.uni-koblenz.de/2008/4/comm-kat.owl#sv1

  ClassAssertion(sv1 soccer-video-data)
  ObjectPropertyAssertion(extended-dns-very-lite:plays sv1 adr2)
200 // Individual: http://isweb.uni-koblenz.de/2008/4/comm-kat.owl#sa2

  ClassAssertion(sa2 core:semantic-annotation)
  ObjectPropertyAssertion(extended-dns-very-lite:setting-for sa2 vd1)
  ObjectPropertyAssertion(extended-dns-very-lite:satisfies sa2 alg1)
205 ObjectPropertyAssertion(extended-dns-very-lite:setting-for sa2 h1)
  // Individual: http://isweb.uni-koblenz.de/2008/4/comm-kat.owl#vd1

  ClassAssertion(vd1 core:video-data)
  ObjectPropertyAssertion(extended-dns-very-lite:plays vd1 adr1)
210 // Individual: http://isweb.uni-koblenz.de/2008/4/comm-kat.owl#sa4

  ClassAssertion(sa4 core:semantic-annotation)
  ObjectPropertyAssertion(extended-dns-very-lite:setting-for sa4 h1)
  ObjectPropertyAssertion(extended-dns-very-lite:setting-for sa4 sv1)
215 ObjectPropertyAssertion(extended-dns-very-lite:satisfies sa4 alg2)
  // Individual: http://isweb.uni-koblenz.de/2008/4/comm-kat.owl#sa1

  ClassAssertion(sa1 core:semantic-annotation)
  ObjectPropertyAssertion(extended-dns-very-lite:setting-for sa1 vd1)
220 ObjectPropertyAssertion(extended-dns-very-lite:satisfies sa1 alg1)
  ObjectPropertyAssertion(extended-dns-very-lite:setting-for sa1 j1)
  // Individual: http://isweb.uni-koblenz.de/2008/4/comm-kat.owl#adr1

  ClassAssertion(adr1 core:annotated-data-role)
225 // Individual: http://isweb.uni-koblenz.de/2008/4/comm-kat.owl#alg1

  ClassAssertion(alg1 core:algorithm)
  ObjectPropertyAssertion(extended-dns-very-lite:defines alg1 slr1)
  ObjectPropertyAssertion(extended-dns-very-lite:defines alg1 adr1)
230 // Individual: http://isweb.uni-koblenz.de/2008/4/comm-kat.owl#sa3

  ClassAssertion(sa3 core:semantic-annotation)
  ObjectPropertyAssertion(extended-dns-very-lite:satisfies sa3 alg2)
  ObjectPropertyAssertion(extended-dns-very-lite:setting-for sa3 j1)
235 ObjectPropertyAssertion(extended-dns-very-lite:setting-for sa3 sv1)
  // Individual: http://isweb.uni-koblenz.de/2008/4/comm-kat.owl#adr2

  ClassAssertion(adr2 core:annotated-data-role) )
```