Multimedia Tools and Applications (2021) 80:6959–6989
https://doi.org/10.1007/s11042-020-09772-y

Creating explorable extended reality environments with semantic annotations

Jakub Flotyński[1]

Received: 26 February 2020 / Revised: 23 July 2020 / Accepted: 28 August 2020 /
Published online: 23 October 2020

Abstract
The main element of extended reality (XR) environments is behavior-rich 3D content consisting of objects that act and interact with one another as well as with users. Such actions and interactions constitute the evolution of the content over time. Multiple application domains of XR, e.g., education, training, marketing, merchandising, and design, could benefit from the analysis of 3D content changes based on general or domain knowledge comprehensible to average users or domain experts. Such analysis can be intended, in particular, to monitor, comprehend, examine, and control XR environments as well as users' skills, experience, interests and preferences, and XR objects' features. However, it is difficult to achieve as long as XR environments are developed with methods and tools that focus on programming and 3D modeling rather than expressing domain knowledge accompanying content users and objects, and their behavior. The main contribution of this paper is an approach to creating explorable knowledge-based XR environments with semantic annotations. The approach combines description logics with aspect-oriented programming, which enables knowledge representation in an arbitrary domain as well as transformation of available environments with minimal users' effort. We have implemented the approach using well-established development tools and exemplify it with an explorable immersive car showroom. The approach enables efficient creation of explorable XR environments and knowledge acquisition from XR.

Keywords Extended reality · 3D Web · Exploration · Reasoning · Queries · Semantic web · Ontologies · Annotations

1 Introduction

Extended reality (XR) covers different forms of combined real and virtual environments, ranging from augmented reality (AR) to virtual reality (VR) in the reality-virtuality continuum [34], and encompassing different types of presentation and interaction with objects in

✉ Jakub Flotyński
flotynski@kti.ue.poznan.pl

[1] Poznań University of Economics and Business, Niepodległości 10, 61-875, Poznań, Poland

such environments [27]. Realistic immersion in XR has been enabled by the increasing performance of GPUs as well as the variety of available headsets and controllers. XR attracts users in multiple application domains, such as marketing, merchandising, simulation, design, engineering, medicine, education, and training. A key element of XR environments is behavior-rich 3D content whose objects perform actions and interactions with one another and with content users, leading to the creation, modification, and destruction of objects in 3D scenes.

Multiple application domains can benefit from registering behavior of users and 3D content in XR, including their actions and interactions described using general or domain knowledge. Registered behavior can be subject to exploration with queries about 3D content states at different moments and periods in time. It may be especially useful in XR intended to acquire knowledge about the environment behavior as well as users' behavior, experience, interests, and preferences. For example, in design, collaborating team members can register and query about the history of a project, changes introduced by the particular designers, and consistency between the changes and the requirements for the project. In marketing and merchandising, collected information about actions of customers interacting with products and salesmen in virtual stores can provide knowledge of their interests and preferences. This, in turn, can help in the proper arrangement of real stores as well as presentation of personalized offers to the customers. In engineering, collected information about the states and behavior of running machines can serve to analyze how the machines work and to identify their possible faults. In medicine, collected information about the steps of virtual treatments can be analyzed by students. In general, information collected while training can be used to consider diverse situations and teach beginners.

However, the aforementioned cases still go beyond the possibilities of the available methods and tools, including 3D formats, programming languages, 3D modeling environments, and game engines, which have not been intended for creation of explorable XR environments. In particular:

1. The available solutions focus on 3D content objects, properties, animations and interactions encoded in a way intelligible to programmers and graphics designers, and interpretable by 3D browsers, rather than general or domain-specific objects, properties, actions and interactions intelligible to average users or experts in the field. This is a problem since XR environments in different application domains are typically used by people with expertise in fields different from IT, who are familiar with general or domain knowledge but not technical and programming concepts.
2. Metadata frameworks in the available 3D formats and tools, such as Extensible 3D (X3D), 3ds Max and Unity, are general and offer no strict formalism of classes, individuals, properties as well as relations between them. This limits their expressiveness and accuracy, thereby narrowing the possibilities of practical applications. Furthermore, unstructured or semi-structured metadata is unsuitable for complex and precise reasoning, which leads to the inference of implicit (tacit) content properties based on explicit properties, and queries including various conditions on content properties.
3. The problem remains even in the case of approaches that enable implementation of 3D content in an object-oriented way. The solutions of this group are improper for knowledge exploration including automated reasoning and query processing, which require the explicit implementation of additional functions. Such implementation is typically done using the same technology as the technology used to build the primary version of the environment. Special effort is needed to decouple the new functions responsible for knowledge processing from the original functions responsible for processing of

3D content, e.g., by refactoring the former code, applying appropriate design patterns, and implementing new software modules. It may be time-consuming and expensive, especially in case of extending existing environments by programmers who have not developed them.

4. Methods of tracking and registering behavior of XR environments should be flexible and efficient in terms of covering different elements of the environment, e.g., modules, classes, functions, and variables, with simple selection and change of target elements, and without introducing redundancy to the original code.

The main contribution of this paper is an approach to creating explorable XR environments. The approach enables both: creation of explorable XR environments from scratch as well as transformation of existing environments to their explorable counterparts. The approach consists of three main interrelated elements: a new formal model of XR behavior, a method of creating explorable XR environments, and an algorithm of generating XR behavior logs, which enable exploration. The general idea of the approach is depicted in Fig. 1. It applies declarative *annotations* to code written in an imperative programming language in the aspect-oriented fashion. The subject of annotation may be the code of any XR environment, in particular, developed using a game engine. Due to the use of the aspect-oriented approach, we minimize the necessary modifications of the available environment code. An annotated environment uses the algorithm we propose to generate *semantic behavior logs* while being used. Behavior logs rely on the new temporal *representation of 3D content behavior* that is built upon the semantic web approach. Behavior logs include information about users' and objects' actions and interactions as well as their results, covering the creation, modification, and destruction of 3D objects and properties. Due to the use of the semantic web, behavior logs can be subject to exploration based on reasoning and queries.

The semantic web is one of the predominant approaches to knowledge representation in different domains and one of the main trends in the evolution of the web [4]. It gains increasing attention in the context of graphical systems and XR, e.g., for photogrammetry [3], molecular visualization [46, 47], content description and retrieval [43, 44], design of industrial spaces [38], archaeology [13] as well as feature-based data exchange between heterogeneous CAD systems [53, 60]. Due to the use of the semantic web in our method, XR behavior can be represented with general or domain knowledge, thus being intelligible to average users and domain experts who are not IT-specialists. Second, the generated behavior logs can be processed with standard reasoning engines to infer implicit knowledge based on explicit knowledge. This allows users who develop or transform an environment to put stress only on its fundamental objects and properties, and skip objects and properties that can be inferred while reasoning. Next, both the explicit and implicit past and current objects and properties can be subject to exploration with queries. Furthermore, due to ontologies, which have been intended as shared terminological and assertional assets, the method can

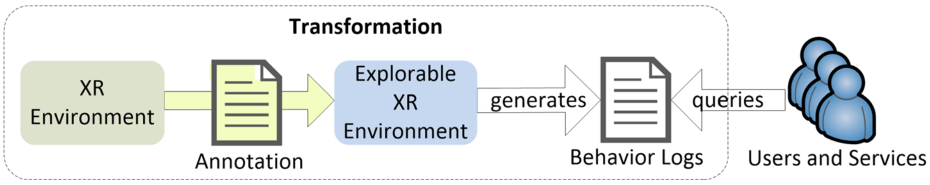


Fig. 1 The concept of annotation-based creation of explorable XR environments

be used in collaborative XR environments, in which multiple users commonly create 3D content based on shared conceptualization.

The remainder of this paper is structured as follows. Section 2 provides an overview of the current state of the art in the fields related to the paper. The proposed method is explained in Section 5. In Section 4, we overview the semantic behavior representation, which is a key element of the method. Section 6 outlines the algorithm used in the method, which is responsible for generating behavior logs compliant with the representation. In Section 7, we describe the tool for development of explorable XR environments, which implements our approach. Further, we illustrate the concepts and toll with an explorable immersive car showroom (Section 8). It is followed by the evaluation (Section 9) and discussion of the results (Section 10). Finally, Section 11 concludes the paper and indicates possible future research.

2 Related work

2.1 Modeling behavior of 3D content

3D content behavior covers actions performed by 3D objects, interactions between objects as well as interactions between objects and users. Both actions and interactions are typically reflected by animations of objects' geometry, structure and appearance.

A number of 3D modeling tools and game engines enable implementation of interactions and animations of 3D content, including 3D modeling tools (e.g., 3ds Max [1] and Blender [5]), animation modeling tools (e.g., Motion Builder [2]) as well as game engines (e.g., Unity [45] and Unreal [23]). The tools permit modeling of interactions and animations covering diverse content features. In addition, some of them, e.g., Blender and Unity, allow for scripting in different programming languages (e.g., Python and C#), which is the most powerful way to implement content behavior. Other tools, which do not require programming skills, are state diagrams and keyframes with interpolation of objects' properties, e.g., geometry, position, orientation, and colors.

The tools use multiple 3D content representation formats, e.g., FBX, OBJ and VRML. However, since the formats have been intended strictly for representing 3D content, they enable neither representation of domain-specific semantics of 3D content nor content exploration with reasoning and queries. Even more complex formats, such as the Extensible 3D (X3D) [59], which enable encoding of metadata, have limited expressiveness with no formalism and structures for content description, such as hierarchies of classes and properties as well as relations between them.

2.2 Semantic web

The semantic web is an emerging trend influential to a growing number of systems in different domains. It is currently one of the leading approaches to knowledge representation offering well-established standards with thoroughly investigated computational properties [55]. Content descriptions based on the semantic web are human-readable and computer-processable. Therefore, we have chosen the semantic web to be the foundation of our approach.

The main standards used to represent content of any type on the semantic web are the Resource Description Framework (RDF) [57], the RDF Schema (RDFS) [58] and the Web Ontology Language (OWL) [54]. RDF is a data model based on statements in the form of triples *[subject predicate object]*. In a statement, the *subject* is what we describe, the *predicate* is a property of the subject, and the *object* is a predicate value, a descriptor, or another entity that is in a relationship with the subject. RDFS and OWL are languages built upon RDF, providing higher expressiveness—classes and properties with relations and hierarchies, which enable comprehensive description of content.

These standards permit design of *ontologies*, which are specifications of conceptualization [25] for a domain. Ontologies are *formal conceptualization of the intended semantics of a knowledge domain or common sense human knowledge, i.e. an abstract, simplified view of the world represented for a particular domain* [44]. Ontologies enable knowledge representation, which can be expressed using statements that belong to three groups. *Terminological knowledge* (TBox) describes conceptualization—a set of concepts and relations between them. *Relational knowledge* (RBox) describes hierarchies and properties of relations. *Assertional knowledge* (ABox) describes facts about individuals (objects) using concepts formalized in TBox and RBox. Ontologies may describe arbitrary objects as well as classes and properties of objects, which makes them a general solution for content description across diverse applications and domains. Ontologies constitute the foundation of the semantic web. Ontologies developed with RDF, RDFS and OWL can be queried using SPARQL [56], which is the main query language on the semantic web.

The aforementioned attributes make ontologies a more strict, structured and comprehensive tool for content description than typical metadata based on schemes with properties and keywords.

2.3 Metadata and semantics for 3D content behavior

Several solutions have been devised to express metadata and semantics of interactions and animations. The analyzed solutions are summarized in Table 1.

The MPEG-7 standard [28] offers several descriptors for different features of multimedia content such as colors, textures, shapes, and motion. An overview of the motion descriptors and their applications has been presented in [12]. The descriptors cover motion activities and trajectories, camera motion, and warping. In addition, the standard specifies the XML-based Description Definition Language, which enables creation of descriptors and descriptor schemes.

In [7], an approach to describing metadata for interaction of 3D objects has been presented. The proposed interaction model encompasses events, conditions, and actions. They are encoded using XSD schemes. In addition, a query language about interaction metadata, with syntax similar to SQL, has been proposed. The solution does not address representation of temporal 3D objects and properties nor reasoning over 3D content. In the paper, an example related to interaction description for virtual heritage objects is presented.

Several approaches enable representation and modeling of 3D content behavior with ontologies and semantic web standards, which offer more complex formalism than typical metadata. An extensive review of the state of the art in ontology-based modeling and representation of 3D content has been presented in [20]. In this paper, we overview the most essential aspects of the solutions.

The approach proposed in [8] uses ontologies to build multi-user virtual environments and avatars. The focus is on representing geometry, space, animation, and behavior of 3D content. The covered entities are semantic equivalents of entities incorporated in widely-used 3D formats, such as VRML and X3D. Environmental objects, which are the main entities of 3D content, are described by translation, rotation, and scale. Avatars are described by names, statuses, and UIs. The avatars' behavior is specified by scripts linked to the scene

Table 1 Metadata- and semantics-based representation of 3D behavior

Approach \ Criteria	Metadata (M) / Semantics (S)	Specificity Level (3D / domain)	Example of Use	Flexible 3D-domain Connection
[12] [28]	M	3D	video browsing	✓
[7]	M	3D	virtual museum	–
[8]	S	3D	virtual humans	
[11] [10] [37]	S	3D and domain	game design	✓
[26] [24]	S	domain	virtual humans	–
[29]	S	3D	genome visualization	
[33]	M	domain	rigid body simulation	–
[40]	S	domain	human embryo	✓
[46]	S	domain	molecular visualization	–
[18] [50] [19]	S	3D and domain	virtual museum	✓
[22]	S	3D	cultural heritage	–

using descriptors. However, scripts do not allow for query-based exploration, reasoning, and temporal content representation.

In the approach proposed in [10, 11, 37], primitive and complex objects' behavior can be specified. Examples of primitive behavior are: move, turn, and roll. Primitive behavior may be combined into complex behavior using temporal, lifetime, and conditional operators, e.g., before, meets, overlaps, starts, enable, and disable. Examples of complex behavior are: build a museum and destroy a bridge. In addition, complex behavior can be modeled using diagrams. Like in the previous solution, the implementation underlying the semantic description of behavior is based on scripts, which imposes similar limitations. The implemented behavior can be exported to X3D scenes.

The ontology proposed in [48] contains classes and properties that represent animations using keyframes, which are linked to geometrical and structural descriptions of the objects. The ontology does not allow for temporal content representation.

The ontology of virtual humans [24, 26] consists of geometrical descriptors (describing vertices and polygons), structural descriptors (describing levels of articulations), 3D animations of face and body as well as behavior controllers (animation algorithms). In addition, the approach enables ontology-based representation of emotional body expressions.

The ontology proposed in [29] enables 3D content representation using classes and properties that are equivalents of X3D nodes and attributes, e.g., textures, dimensions, coordinates, and LODs. In addition, the approach permits specification of rules. For instance, if an individual is of the atom class (body), generate a sphere to represent it in an ontology for chemical compounds (head). Thus, final 3D content is based on both the explicit and implicit (inferred) knowledge.

The simulation environment presented in [33] combines a few tools. The Unreal game engine enables rigid-body physics and content presentation. An inference engine enables reasoning and updating the scene representation when events occur. A behavioral engine enables action recognition and changes in conceptual objects' properties. The use of different engines within one environment permits separation of concerns between users with different expertise.

An approach to spatio-temporal reasoning over ontology-based representations of evolving human embryo has been proposed in [40]. The ontologies are encoded in RDF and OWL, and they describe stages, periods, and processes.

In [46], an ontology-based representation of 3D molecular models has been proposed. The approach combines different input (e.g., interaction using various haptic and motion tracking devices) and output (e.g., 2D and 3D presentation) modalities to enable presentation and interaction suitable for different kinds of devices, content, and tasks to be performed.

The approach proposed in [16–18, 49, 50] uses ontologies to represent 3D content at different specificity levels—related to 3D graphics as well as an application domain. Furthermore, it enables reasoning to liberate content authors from determining all content properties. It permits semantic queries to generic 3D templates (meta-scenes) in order to generate customized 3D scenes. Finally, the customized 3D scenes can be transformed into different 3D formats understandable to 3D browsers.

In [22], an approach to generating ontology-based 3D formats from available 3D formats has been proposed. The X3D Ontology [51], which is a semantic equivalent of the Extensible 3D (X3D) [59] format has been presented. It gathers all concepts of X3D, including animation and interaction. The ontology has been automatically generated from the X3D XML Schema combined with the X3D Unified Object Model (X3DUOM), which complements the schema with information about classes of and relationships between X3D nodes.

2.4 4D Fluents

Semantic temporal representation of content has been extensively studied in the domain of the semantic web leading to the development of a few solutions. One of the available approaches is 4D fluents [52]. A *fluent* is a property that varies over time. The approach specifies rules to be accomplished to transform a time-independent statement into a time-dependent statement. The result of the transformation of a statement *two objects are linked by a property* is a statement *two objects are linked by a property at a time point or within a time interval*.

This is achieved by using the concept of *time slices*, which are temporal counterparts to the primary objects, associated with *time entities*. The representation of an object that has several different values of a property in different points or intervals of time includes several distinct time slices of the object (one for every point/interval) that are associated with the points/intervals and are assigned the proper property values.

For instance, to express that a virtual salesman was working in a store from 10 to 11 am., create time slices for both the salesman and the store, and link them by the *worked in* property. Next, create the interval representing that 1 hour, and assign it to the time slices.

3 Problem statement

So far, the semantic representation of behavior, including animations and interactions has gained little attention in comparison to the representation of static content features related to geometry, structure, and appearance. The available approaches in this field are still preliminary and have the following limitations, which also determine the main requirements for our approach:

1. They do not enable representation of temporal 3D content objects and properties with their changes over time.
2. They do not enable flexible, applicable to different domains and contexts, connection between temporal 3D content objects and properties with domain-specific or general objects and properties that are visualized by the 3D content.
3. They do not enable reasoning over temporal 3D content objects and properties, which can lead to the inference of implicit (tacit) knowledge from explicitly specified knowledge. It could allow content authors to focus only on fundamental features of the content while skipping features that are implicated by the fundamentals.
4. They do not enable registration of users' and objects' behavior, including actions and interactions that occur while using XR environments.

4 Semantic behavior representation

The main contribution of this paper is an approach to *annotation-based creation of explorable XR environments*, which covers the shortfalls of the available solutions mentioned in Section 3. The approach consists of two main elements: the *semantic behavior representation*, which is a formal model of XR behavior, and the *method of annotation-based development of XR environments*, which is presented in Section 5.

The *semantic behavior representation* is a pair of ontologies based on the semantic web standards: RDF, RDFS, and OWL. The ontologies permit semantic representation of actions of 3D content objects and users as well as interactions between 3D objects, and between users and 3D objects. In addition, they enable representation of results of actions and interactions. Due to the use of the semantic web, 3D content behavior can be represented using general or domain knowledge. The ontologies of the representation are sets of statements that express TBox and RBox (cf. Section 2.2).

4.1 Behavior representation ontologies

The behavior representation consists of a domain ontology and the fluent ontology, which are presented in the following subsections.

4.1.1 Domain ontology

A *domain ontology* specifies classes and properties related to a particular application domain, comprehensible to average users or domain experts. A domain ontology is determined by a particular XR application, and it is common to all its use cases. Different domain ontologies may be used for behavior representations across different explorable environments, e.g., exhibitions in a virtual museum, buildings in a virtual city, and cars in a virtual showroom.

4.1.2 Fluent ontology

The *fluent ontology* specifies classes and properties based on the 4D-fluents approach (cf. Section 2.4), which describe intervals, points in time and time slices. The classes and properties are used to represent temporal content properties (referred to as *fluents*) that are specified in the domain ontology. The fluent ontology is an immutable part of our approach, and it is common to all explorable XR environments.

The ontology has been specified using a description logic enabling: class intersection and union as well as qualified cardinality restrictions. The ontology is presented in Listing 1 using the notation for description logics [30]. *TimePoint* is the class of time entities that are instant—have duration equal to zero (line 1), while *TimeInterval* is the class of time entities that have duration larger than zero (2). The *start* and *end* properties are specified for time intervals, and indicate time points (3-6). Every time slice has exactly one object indicated using the *isTimeSliceOf* property and exactly one of two properties *hasTimePoint* or *hasTimeInterval*, which indicates a time entity in which the time slice exists (8-9). The domains and ranges of the properties are specified like for *start* and *end* (10-14).

4.2 Behavior logs

A *semantic behavior log* is a set of statements that express *assertional knowledge* (ABox) about behavior-rich domain objects, which are represented by animated and interactive 3D objects of an XR environment, and described using entities specified in the behavior representation, including the fluent and domain ontologies. A behavior log represents a temporal state of affairs in a particular use (session) of an explorable XR environment, while users and 3D objects are acting and interacting. Actions and interactions reported in logs can be further explored by reasoning engines to infer tacit (implicit) users' and objects' properties on the basis of their explicit properties. Moreover, logs can be queried by users, applications, and services to acquire information about involved users and objects, their actions and interactions, added and removed objects as well as modified users' and objects' properties, in different points and periods of time.

1 $TimePoint \sqsubseteq TemporalEntity$
2 $TimeInterval \sqsubseteq TemporalEntity$
3 $\exists start.\top \sqsubseteq TimeInterval$
4 $\top \sqsubseteq \forall start.TimePoint$
5 $\exists end.\top \sqsubseteq TimeInterval$
6 $\top \sqsubseteq \forall end.TimePoint$
7
8 $TimeSlice \sqsubseteq (=1\ \ isTimeSliceOf.\top$
9 $\quad \sqcap\ (=1\ \ hasTimePoint.TimePoint\ \sqcup\ =1\ \ hasTimeInterval.TimeInterval))$
10 $\exists isTimeSliceOf.\top \sqsubseteq TimeSlice$
11 $\exists hasTimePoint.\top \sqsubseteq TimeSlice$
12 $\top \sqsubseteq \forall hasTimePoint.TimePoint$
13 $\exists hasTimeInterval.\top \sqsubseteq TimeSlice$
14 $\top \sqsubseteq \forall hasTimeInterval.TimeInterval$

Listing 1 Fluent ontology for the XR behavior representation

5 Method of annotation-based development of explorable XR environments

The concept of the method, which is the second key element of the approach, is depicted in Fig. 1. An *explorable XR environment* is an XR environment in which selected users' and objects' properties are logged using general or domain knowledge, while the environment is being used. The registration, also referred to as *logging*, covers changes of users' and objects' properties over time. Properties are registered in the form of *semantic behavior logs*, which comply with the *semantic behavior representation* (cf. Section 4).

The goal of the method is to enable creation of explorable XR environments by involving domain experts who have knowledge in the particular application domain but have no advanced technical skills. To achieve this, we provide the method, which liberates the authors from regularly programming the environment code in favor of annotating it using domain knowledge. Therefore, the method is based on aspect-oriented programming in combination with the semantic web approach, which permits creation of new explorable environments from scratch as well as transformation of existing environments to their explorable counterparts. The method is intended to minimize changes to the 3D content management layer, which could affect the presentation and behavior of the environment as an undesirable side effect. Finally, it separates the knowledge management layer, which is extra to the environment, from the 3D content management layer.

The successive steps of the method are depicted in Fig. 2. It the steps, the code of a prototype XR environment is annotated and compiled, leading to the generation of an explorable XR environment. The method significantly extends the approach proposed in [15, 21] with representing autonomous actions of 3D content objects in a way that is independent of XR ontologies.

5.1 Step 1: Annotating the prototype environment

A *prototype XR environment*, which is given at the input of the method, is an application implemented using a game engine (e.g., Unreal or Unity), an imperative programming language (e.g., Java, JavaScript, C++ or C#) and libraries for 3D/XR development (e.g., Java 3D, WebGL, and DirectX). A prototype environment is encoded in a procedural or object-oriented way, thus it consists of functions or classes with methods.

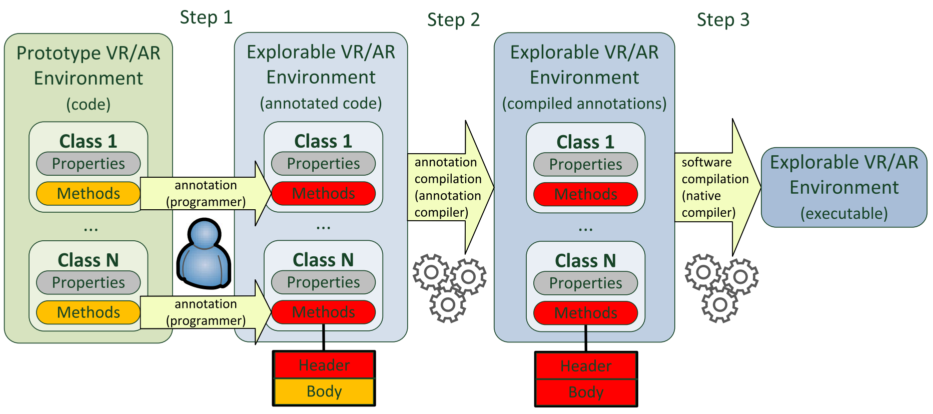

Step 1
Step 2
Step 3
Prototype VR/AR Environment (code)
Class 1
Properties
Methods
...
Class N
Properties
Methods
annotation (programmer)
annotation (programmer)
Explorable VR/AR Environment (annotated code)
Class 1
Properties
Methods
...
Class N
Properties
Methods
Header
Body
annotation compilation (annotation compiler)
Explorable VR/AR Environment (compiled annotations)
Class 1
Properties
Methods
...
Class N
Properties
Methods
Header
Body
software compilation (native compiler)
Explorable VR/AR Environment (executable)

Fig. 2 Transforming XR environments to explorable XR environments

5.1.1 Selection of class methods for annotation

In this step, a designer annotates some functions or class methods (depending on the programming paradigm used) in the prototype environment. A designer may be a domain expert with extensive knowledge of the application domain and basic knowledge of the hierarchy of classes and methods in the project. The selection of class methods is arbitrary. It depends on the desirable information that should be included in behavior logs that will be generated while using the environment. In general, subject to annotation are class methods, whose successful execution has to be reported in the behavior logs. For instance, the annotation of a class method that puts a product into the shopping cart allows marketing specialists to study consumers' behavior in a virtual store. The annotation of a class method that turns on appliances on a power station allows new employees to learn how to use the equipment. The annotation of class methods that add and modify buildings in a 3D urban area allows architects to follow the city design process. Apart from generating fragments of behavior logs, annotated class methods behave in the same way as their equivalents in the prototype XR environment. The result of this step is annotated code, in which the headers of the selected class methods are preceded by annotations. In this step, the bodies of the class methods remain unchanged.

5.1.2 Semantics and syntax of annotations

In our approach, every class method may be preceded by multiple annotations, e.g., in Listing 2. Every annotation consists of at least three parameters: a subject, a predicate, and an object, which form an RDF statement (cf. Section 2.2). The optional fourth parameter is a timestamp that is a moment or a period when the statement is true. In an annotation, a subject, predicate and object can be given as literal values, parameters of the class method, local variables, or global variables.

Different annotations of the same or different class methods may use common entities: subjects, predicates, objects, and timestamps. The same entities used in annotations attached to a common method denote the same values. Also, the same entities that indicate global variables and are attached to different methods denote the same values. Otherwise, the values are determined by the indicated method parameters or local method variables.

In the example in Listing 2, since the method belongs to the `Visitor` class, the `this` keyword in the first annotation indicates the subject as the visitor for whom the method is invoked. The predicate is the `watches` property from a domain ontology for virtual galleries. `Painting` indicates a method parameter, which is used as the object of the first annotation. The action of watching the painting by the visitor will be reported as occurring since the current moment as indicated by the `start` timestamp. In a while, the visitor may be watching another painting. To collect more comprehensive information about the visitor's action, also the author of the painting, his/her name, and birth date are registered by the other three annotations. The annotations use predicates specified in the Dublin Core [9], FOAF [6] and DBpedia [32] ontologies, and indicate objects represented by variables in the method body. Since the annotations describe immutable data, they include no timestamps. Successful execution of the method adds semantic statements derived from the four annotations to a behavior log in a triplestore. Information collected in behavior logs generated using this annotated method allows for analyzing visitors' interests in art, including their favorite periods and artists.

```
[SemanticLog("this", "gallery:watches", "painting", "start")]
[SemanticLog("painting", "dc:creator", "author")]
[SemanticLog("author", "foaf:name", "name")]
[SemanticLog("author", "dbpedia:birthDate", "birthDate")]
public void Watch(Painting painting) {
  WatchedPaintings.Add(painting);
  Person author = painting.Author;
  string name = author.Name;
  Date birthDate = author.BirthDate;
  //the rest of method instructions ... }
```

Listing 2 Example of an annotated class method in C#

5.2 Step 2: Compiling the annotations

In this step, the annotation compiler that we have implemented (cf. Section 7) processes the entire project code and completes the following actions.

1. It attaches the *semantic log library* to the project. The library implements the algorithm for generating logs (cf. Section 6). It derives statements from annotations and loads them to a triplestore.
2. It extends the body of every annotated class method with appropriate instructions that generate log statements while the environment is being used. Processing each individual annotation of a class method injects new instructions that are responsible for generating logs to the end of the method. The scheme of a generated class method is presented in Listing 3. The added instructions are responsible for extracting information about the classes of the variables used in the annotations, creating timestamps, setting identifiers of the semantic individuals to be inserted to the log, and invoking a method from the library that inserts the statements with the individuals to the triplestore. The input parameters of the method are the variable values and literal values given in the annotations as well as the extracted classes and created timestamps.
3. It injects additional common classes and methods responsible for setting values of temporary variables used in the code inserted in action 2.

Listing 3 Example of a compiled annotated class method in C#

```
public void Watch(Painting painting) {
  WatchedPaintings.Add(painting);
  Person author = painting.Author;
  string name = author.Name;
  Date birthDate = author.BirthDate;
  ...
  //generated instructions for logging
}
```

5.3 Step 3: Compiling the environment

In this step, a native compiler specific for particular hardware and software platform is used to generate the explorable XR environment in its executable form, which is the result of the method. The explorable XR environment comprises the same classes with the same properties and methods as the prototype environment. Hence, its 3D objects appear and behave like their prototypes. However, the class methods with annotations created in Step 1 and compiled in Step 2 generate behavior logs while the environment is used. Behavior logs conform to the semantic behavior representation presented in Section 4.

6 Algorithm for generating behavior logs

In addition to the semantic behavior representation, an essential element used by the method is the algorithm for generating semantic behavior logs, which conform to the representation. The implementation of the algorithm is injected into every explorable XR environment in Step 2 of the method (cf. Section 5.2), and it is running during a session of using the environment. The algorithm registers actions as well as interactions occurring in the environment that are executed by class methods annotated in Step 1 of the method (cf. Section 5.1). The algorithm has the following steps for every annotated class method:

1. For every annotation, determine the values of the `subject`, `predicate` and `object` in the following order.
   (a) If a `subject/predicate/object` is enclosed in quotation, it is processed as the literal value.
   (b) If a `subject/predicate/object` is not enclosed in quotation:
      i and is equal to the identifier of a method parameter, it is processed as the value of the parameter.
      ii and is equal to the identifier of a method variable, it is processed as the value of the variable.
      iii and is equal to the identifier of a global variable, it is processed as the value of the variable.

   Comment. *A subject, predicate, and object of an annotation may indicate either a literal value, method parameter, local method variable, or global class variable. Quotation indicates that the subject/predicate/object is a literal value. Otherwise, the priority of variables is determined by points 1(b)i-1(b)iii.*
2. For every annotation without a `timestamp`, which has the form `[SemanticLog( subject, predicate, object)]`, insert the statement `<subject predicate object>` to the triplestore.
   Comment. *Processing of annotations without timestamps is simpler as it does not cover creation of time slices, time points nor time intervals.*
3. For all annotations with a `timestamp`, which have the form `[SemanticLog( subject, predicate, object, timestamp)]` where `timestamp` does not indicate a particular date, determine the common value equal to the current date, which denotes:
   (a) a time point, if the `timestamp` is equal to `now`.
   (b) the beginning of a time interval, if the `timestamp` is equal to `start`.

(c) the end of a time interval, if the `timestamp` is equal to `end`.

Comment. *Annotations with timestamps set to exact dates will use the dates to express temporal statements. Annotations with timestamps set to keywords must be commonly processed—to generate statements that refer to the same points or intervals in time.*

4. For every annotation with a `timestamp`:

   (a) If the `timestamp` is equal to `end` or the `predicate` is a functional property, update the open time interval (with no end property set) associated with a subject `time slice` and the `predicate`, if such one exists, by setting its end property to the `timestamp` value.

   Comment. *An interval in which a subject has a particular predicate may be closed explicitly, by using the end keyword with an annotation related to this subject and predicate. If the predicate is a functional property, which may have at most one value at a given time for a particular subject, a similar effect is caused by setting a new value for the predicate assigned to the subject.*

   (b) If the `timestamp` is equal to `now` or `start`:

   i If the `timestamp` is equal to `now`, create a `time point`, which is an individual of the `TimePoint` class, and set its value property to the determined `timestamp` value.

   Comment. *The now keyword indicates the current moment in time considered as a time point.*

   ii If the `timestamp` is equal to `start`, create a `time interval`, which is an individual of the `TimeInterval` class, and set its start property to the determined `timestamp` value.

   Comment. *The start keyword indicates the current moment in time considered as the beginning of a time interval.*

   iii Create a `time slice` of the `subject`, which is an individual of the `TimeSlice` class from the fluent ontology as well as of all the classes of the `subject`.

   iv Assign the created `time point` or `time interval` to the subject `time slice` using the `hasTimePoint` or `hasTimeInterval`, respectively.

   Comment. *The time slice will be a temporal representation of the subject with the particular predicate assigned at a given moment or for a period in time. Therefore, it must link the information about time.*

   v If the predicate is a datatype property, set the `predicate` for the subject `time slice` to the value specified by the object.

   vi If the `predicate` is an object property:

   A Create a `time slice` of the object, which is an individual of the `TimeSlice` class from the fluent ontology as well as of all the classes of the `object`.

   B Set the `predicate` for the subject `time slice` to the object `time slice`.

   C Assign the created `time point` or `time interval` to the object `time slice` using the `hasTimePoint` or `hasTimeInterval`, respectively.

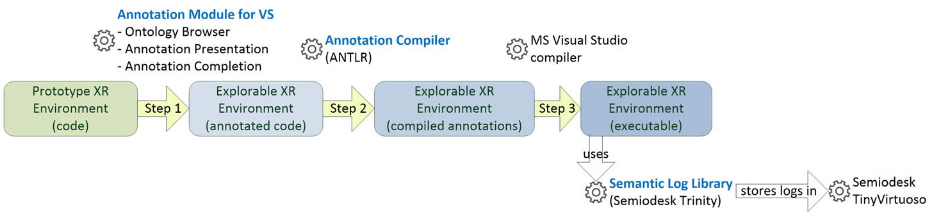


Fig. 3 Architecture and workflow of the *Development Tool for Explorable XR Environments*

Comment. *If the predicate indicates an object, temporal representation must be specified for the object as it was specified for the subject in points 4(b)iii-4(b)iv.*

7 Development tool for explorable XR environments

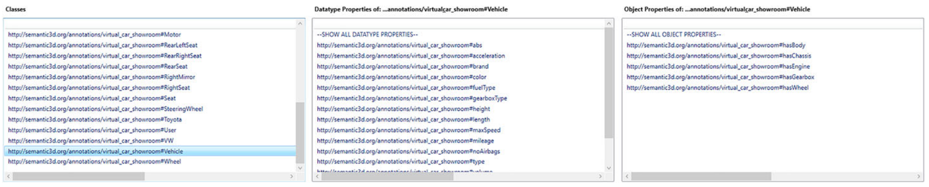
We have implemented the *Development Tool for Explorable XR Environments* to illustrate and evaluate the proposed approach. The tool consists of three modules: the *annotation module*, the *semantic log library* and the *annotation compiler*, which are described in the following subsections. The scheme of the architecture and the workflow of the tool are depicted in Fig. 3. The annotation module supports Step 1 of the method by providing an ontology browser and enabling presentation and completion of annotations. The annotation compiler enables Step 2 of the method, while the semantic log library permits logging of behavior of executable environments.

7.1 Annotation module

The *annotation module* is an extension to the MS Visual Studio IDE. The module offers three functions: ontology browser, annotation presentation, and annotation completion.

7.1.1 Ontology browser

The ontology browser enables loading an arbitrary OWL ontology, whose classes, datatype properties and object properties are presented in three neighboring list views (Fig. 4). The left list view presents all classes specified in the ontology. The central and right list views present all datatype and object properties specified in the ontology, or properties whose


Classes

```
http://semantic3d.org/annotations/virtual_car_showroom#Motor
http://semantic3d.org/annotations/virtual_car_showroom#RearLeftSeat
http://semantic3d.org/annotations/virtual_car_showroom#RearRightSeat
http://semantic3d.org/annotations/virtual_car_showroom#RearSeat
http://semantic3d.org/annotations/virtual_car_showroom#RightMirror
http://semantic3d.org/annotations/virtual_car_showroom#RightSeat
http://semantic3d.org/annotations/virtual_car_showroom#Seat
http://semantic3d.org/annotations/virtual_car_showroom#SteeringWheel
http://semantic3d.org/annotations/virtual_car_showroom#Toyota
http://semantic3d.org/annotations/virtual_car_showroom#User
http://semantic3d.org/annotations/virtual_car_showroom#VW
http://semantic3d.org/annotations/virtual_car_showroom#Vehicle
http://semantic3d.org/annotations/virtual_car_showroom#Wheel
```

Datatype Properties of: ...annotations/virtualçar_showroom#Vehicle

```
--SHOW ALL DATATYPE PROPERTIES--
http://semantic3d.org/annotations/virtual_car_showroom#abs
http://semantic3d.org/annotations/virtual_car_showroom#acceleration
http://semantic3d.org/annotations/virtual_car_showroom#brand
http://semantic3d.org/annotations/virtual_car_showroom#color
http://semantic3d.org/annotations/virtual_car_showroom#fuelType
http://semantic3d.org/annotations/virtual_car_showroom#gearboxType
http://semantic3d.org/annotations/virtual_car_showroom#height
http://semantic3d.org/annotations/virtual_car_showroom#length
http://semantic3d.org/annotations/virtual_car_showroom#maxSpeed
http://semantic3d.org/annotations/virtual_car_showroom#mileage
http://semantic3d.org/annotations/virtual_car_showroom#noAirbags
http://semantic3d.org/annotations/virtual_car_showroom#type
```

Object Properties of: ...annotations/virtualçar_showroom#Vehicle

```
--SHOW ALL OBJECT PROPERTIES--
http://semantic3d.org/annotations/virtual_car_showroom#hasBody
http://semantic3d.org/annotations/virtual_car_showroom#hasChassis
http://semantic3d.org/annotations/virtual_car_showroom#hasEngine
http://semantic3d.org/annotations/virtual_car_showroom#hasGearbox
http://semantic3d.org/annotations/virtual_car_showroom#hasWheel
```

Fig. 4 *Ontology Browser* of the *Development Tool for Explorable XR Environments*

domain is the class selected in the left list view. The properties are used by the other function of the module responsible for presentation and completion of annotations. The fluent ontology (cf. Section 4.1.2) has been implemented using OWL 2 DL.

7.1.2 Annotation presentation

Semantic annotations are instances of as the SemanticLog class, which inherits from the `System.Attribute` class. The SemanticLog class has four fields corresponding to elements of an annotation: subject, predicate, object, and timestamp. Nonetheless, it accepts annotations of two forms—with and without timestamps. The SemanticLog class is added to the code project of an XR environment by the annotation designer in Step 1 of the method (cf. Section 5.1). It enables Visual Studio to validate the syntax of the created annotations.

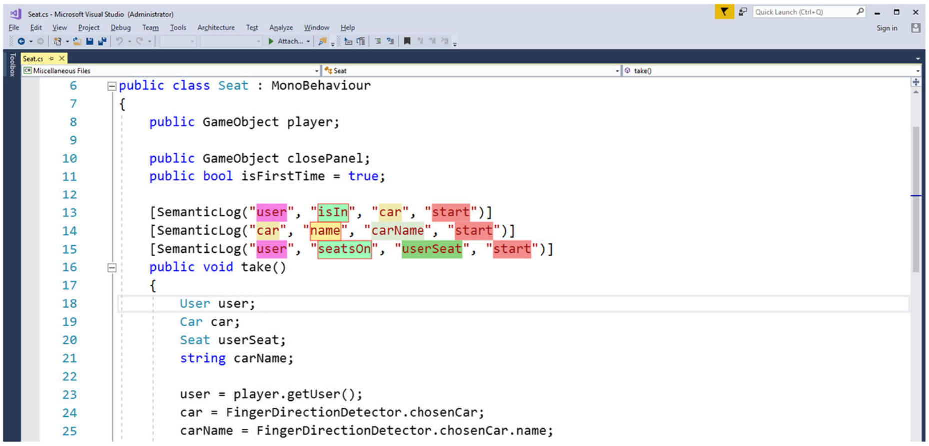
While annotations are programmed, the annotation presentation function paints their subjects, predicates, objects and timestamps using distinct colors. An example of coloring annotations is depicted in Fig. 5. The same subjects and objects have the same colors across different annotations, e.g., `user` and `car`, to denote that they refer to the same variables in the code. In addition, datatype properties (e.g., `name`) are distinguished from object properties (e.g., `isIn` and `seatsOn`).

7.1.3 Annotation completion

The annotation completion function provides suggestions while writing annotations. The suggestions are contextual and depend on what is currently being written in the text area. In Fig. 6a-d, suggested are, respectively, the entire annotation template, subject (from the list of method parameters and variables), predicate (from the list of properties in the ontology), and timestamp (dates and keywords).

7.2 Annotation compiler

The *annotation compiler* is a console application encoded in C#. The input parameters of the compiler are: the path of a prototype XR environment to be transformed and the path of


Seat.cs - Microsoft Visual Studio (Administrator)
Quick Launch (Ctrl+Q)
File Edit View Project Debug Team Tools Architecture Test Analyze Window Help
Sign in
Attach...
Toolbox
Seat.cs
Miscellaneous Files
Seat
take()

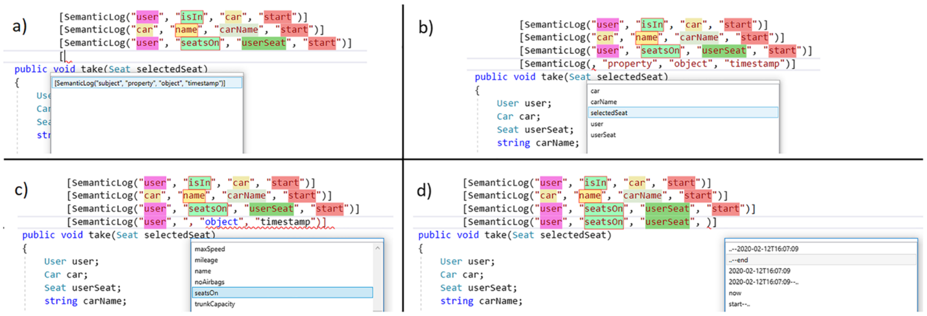
```
public class Seat : MonoBehaviour
{
    public GameObject player;

    public GameObject closePanel;
    public bool isFirstTime = true;

    [SemanticLog("user", "isIn", "car", "start")]
    [SemanticLog("car", "name", "carName", "start")]
    [SemanticLog("user", "seatsOn", "userSeat", "start")]
    public void take()
    {
        User user;
        Car car;
        Seat userSeat;
        string carName;

        user = player.getUser();
        car = FingerDirectionDetector.chosenCar;
        carName = FingerDirectionDetector.chosenCar.name;
```

Fig. 5 Painting annotations in the *Development Tool for Explorable XR Environments*



a)

```
    [SemanticLog("user", "isIn", "car", "start")]
    [SemanticLog("car", "name", "carName", "start")]
    [SemanticLog("user", "seatsOn", "userSeat", "start")]
    [
public void take(Seat selectedSeat)
{
    Use
    Car
    Sea
    str
```

b)

```
    [SemanticLog("user", "isIn", "car", "start")]
    [SemanticLog("car", "name", "carName", "start")]
    [SemanticLog("user", "seatsOn", "userSeat", "start")]
    [SemanticLog(, "property", "object", "timestamp")]
public void take(Seat selectedSeat)
{
    User user;
    Car car;
    Seat userSeat;
    string carName;
```

c)

```
    [SemanticLog("user", "isIn", "car", "start")]
    [SemanticLog("car", "name", "carName", "start")]
    [SemanticLog("user", "seatsOn", "userSeat", "start")]
    [SemanticLog("user", ", "object", "timestamp")]
public void take(Seat selectedSeat)
{
    User user;
    Car car;
    Seat userSeat;
    string carName;
```

d)

```
    [SemanticLog("user", "isIn", "car", "start")]
    [SemanticLog("car", "name", "carName", "start")]
    [SemanticLog("user", "seatsOn", "userSeat", "start")]
    [SemanticLog("user", "seatsOn", "userSeat", )]
public void take(Seat selectedSeat)
{
    User user;
    Car car;
    Seat userSeat;
    string carName;
```

Fig. 6 Annotation completion in the *Development Tool for Explorable XR Environments*

the explorable XR environment to be generated. The compiler implements code injection according to the aspect-oriented approach. It works in Step 2 of the method. It attaches the semantic log library to the project, extends the bodies of annotated methods with invocations of the library methods, and injects additional auxiliary classes that prepare the invocations.

The compiler is based on ANTLR [36], which is a well-established library for creating parsers and compilers. ANTLR has been used to create a grammar, which recognizes method annotations with their parameters and variables to generate the final code.

7.3 Semantic log library

The *semantic log library* is the module responsible for generating semantic behavior logs while an explorable environment is being used. It creates semantic statements on the basis of method annotations. The library has been implemented as a dynamic-link library (DLL) in Visual Studio. Its size is equal to 46 KB. Semantic statements are created using Semiodesk Trinity [42], an open project for ontology engineering in C# applications. The generated logs are stored in Semiodesk TinyVirtuoso [41], which is a triplestore with an HTTP-based interface. The library is attached to the XR environment in Step 2 of the method (cf. Section 5.2) by the annotation compiler. The library is compatible with C# projects. In the examples presented in the rest of this paper, we use the library for projects developed in the Unity game engine.

8 Explorable immersive car showroom

An immersive car showroom has been developed in the virtual reality laboratory at the Poznań University of Economics and Business in Poland [35, 39].

8.1 Design

The showroom is a Unity-based application, which uses an Oculus Rift headset [14] and a Leap Motion hand tracking device [31]. The Unity game engine [45] is a widely-used tool for creating XR applications with different devices such as headsets and motion tracking systems. Oculus Rift enables immersive presentation of 3D content, while the Leap Motion enables interaction with 3D cars using hand gestures.

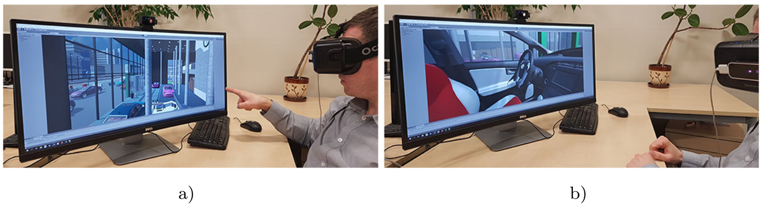


Fig. 7 Selecting a car (**a**) and sitting in the car (**b**) in the immersive car showroom. ©Jakub Flotyński 2020, all rights reserved.

We have annotated and transformed the showroom to an explorable VR environment. An example of a method annotation is depicted in Fig. 5. In the showroom, a user can accomplish the following actions implemented by distinct class methods, using hand gestures.

1. *Select a car*, which is executed by indicating the car using the forefinger of the right hand (Fig. 7a). The user is immediately moved to the car and can watch it from outside. In the showroom, a number of different cars are accessible.
2. *Change the color of the car*, which is executed for the currently watched car by selecting a color from a palette in the main menu using the forefinger of the right hand. The selected color is immediately applied to the body of the car.
3. *Watch a car from around*, which is executed by selecting a direction of rotation around the car from the main menu using the forefinger of the right hand. In this way, the user can view the car from all sides.
4. *Take a seat in the car*, which is executed by selecting a seat inside the car from the main menu. Once got in, the user can watch the car inside (Fig. 7b).

The aforementioned class methods have been annotated using the annotation module of the development tool for explorable XR environments in Step 1 of the method. In addition to the methods, constructors of the Customer and Car classes have been annotated to register information about basic objects for time slices. Next, the annotations have been compiled by the annotation compiler in Step 2. Finally, in Step 3, MS Visual Studio has been used to compile the overall environment to its executable explorable form.

8.2 Example of a behavior log

A behavior log has been generated while a customer was visiting the explorable immersive car showroom. A part of the visit is described in Listing 4. The log presents past behavior of the customer and the environment. The log consists of statements that describe actions and interactions between the customer and cars in the environment, which are implemented by the annotated methods listed in Section 8.1.

Every object in the log is represented by an OWL named individual. Objects are described in a general way understandable to average users. The underlying 3D representation of the objects is irrelevant to the log, as its target users are not necessarily familiar with computer graphics. The prefixes `fo` and `do` pertain to the fluent and domain ontologies, which are the parts of the behavior representation (cf. Section 4). The names of the individuals are distinct due to postfixes, which are combinations of date, time, and hash codes.

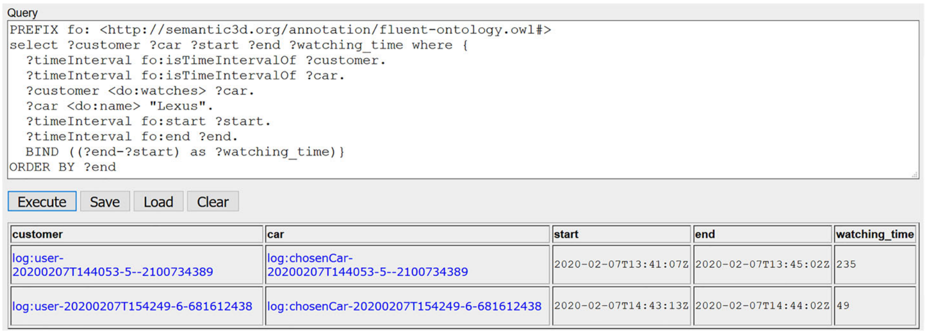
```
 1 log:customer-20200207T144053
 2   rdf:type  owl:NamedIndividual , do:Customer ;
 3   foaf:name "John Kowalski" .
 4
 5 log:car-20200207T144053
 6   rdf:type  owl:NamedIndividual , do:Car ;
 7   do:name  "Volvo" .
 8
 9 log:customer-20200207T144053-1-1775502135
10   rdf:type  owl:NamedIndividual , fo:TimeSlice , do:Customer ;
11   fo:isTimeSliceOf  log:customer-20200207T144053 ;
12   do:watches  log:car-20200207T144053-1-1775502135 .
13
14 log:car-20200207T144053-1-1775502135
15   fo:isTimeSliceOf  log:car-20200207T144053 .
16
17 log:timeinterval-20200207T144053-1-1775502135
18   fo:isTimeIntervalOf  log:car-20200207T144053-1-1775502135 ,
         log:customer-20200207T144053-1-1775502135 ;
19   ns1:start  "2020-02-07T13:41:06Z"^^xsd:dateTime ;
20   ns1:end  "2020-02-07T13:41:18Z"^^xsd:dateTime .
21
22 log:car-20200207T144053-6-338709773
23   rdf:type  owl:NamedIndividual , fo:TimeSlice , do:Car ;
24   fo:isTimeSliceOf  log:car-20200207T144053 ;
25   do:color  "RGBA(0.392, 0.000, 0.000, 1.000)" .
26
27 log:timeinterval-20200207T144053-6-338709773
28   rdf:type  owl:NamedIndividual , fo:TimeInterval ;
29   fo:isTimeIntervalOf  log:car-20200207T144053-6-338709773 ;
30   fo:start  "2020-02-07T13:41:36Z"^^xsd:dateTime ;
31   fo:end  "2020-02-07T13:41:51Z"^^xsd:dateTime .
```

Listing 4 Example of a behavior log

The customer (lines 1-3) and the car (5-7) are basic classes for time slices, with constant (independent of time) properties such as `name`. The customer interacts with the car by watching it, which is expressed by time slices linked by the `watches` property (9-15). It is lasting from `2020-02-07T13:41:06Z` to `2020-02-07T13:41:18Z` as specified by the time interval linked to both the time slices (17-20). Later, the color of the car was changed by the customer for another interval of time (22-31).

8.3 Use cases of behavior exploration

Semantic exploration of actions and interactions in the immersive car showroom is possible with queries to the generated behavior logs. Queries may be encoded in SPARQL [56], which is the main query language for RDF-based ontologies and knowledge bases. Such exploration can provide information about customers' interests and preferences, which can be further used for marketing and merchandising, especially for presenting personalized



Query

```
PREFIX fo: <http://semantic3d.org/annotation/fluent-ontology.owl#>
select ?customer ?car ?start ?end ?watching_time where {
  ?timeInterval fo:isTimeIntervalOf ?customer.
  ?timeInterval fo:isTimeIntervalOf ?car.
  ?customer <do:watches> ?car.
  ?car <do:name> "Lexus".
  ?timeInterval fo:start ?start.
  ?timeInterval fo:end ?end.
  BIND ((?end-?start) as ?watching_time)}
ORDER BY ?end
```

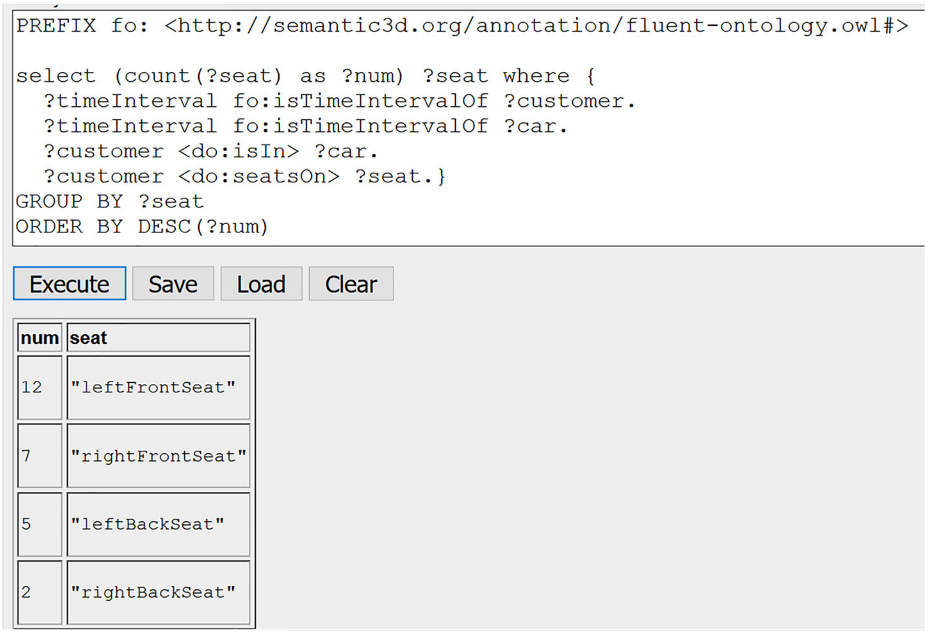
customer	car	start	end	watching_time
log:user-20200207T144053-5--2100734389	log:chosenCar-20200207T144053-5--2100734389	2020-02-07T13:41:07Z	2020-02-07T13:45:02Z	235
log:user-20200207T154249-6-681612438	log:chosenCar-20200207T154249-6-681612438	2020-02-07T14:43:13Z	2020-02-07T14:44:02Z	49

Fig. 8 Behavior exploration query: how much time did a customer spend to watch a car outside?

offers. Examples of an exploration cover the following use cases presented in the Virtuoso web interface.

Query 1: *how much time did the customer spend to watch a particular car outside?* The SPARQL query and its result are presented in Fig. 8. This information can be used to investigate which cars are interesting to customers. In the example, the customer is watching a Lexus car. Customer and car time slices linked to common intervals are searched. Next, the length of the intervals is calculated, and the intervals are presented.

Query 2: *which places inside cars are preferred by the customer?* The SPARQL query and its result are presented in Fig. 9. This information can be used to discover whether the customer is mostly a driver or a passenger. The query looks for seats taken by the customer with no regard to the particular cars.



```
PREFIX fo: <http://semantic3d.org/annotation/fluent-ontology.owl#>

select (count(?seat) as ?num) ?seat where {
  ?timeInterval fo:isTimeIntervalOf ?customer.
  ?timeInterval fo:isTimeIntervalOf ?car.
  ?customer <do:isIn> ?car.
  ?customer <do:seatsOn> ?seat.}
GROUP BY ?seat
ORDER BY DESC(?num)
```

num	seat
12	"leftFrontSeat"
7	"rightFrontSeat"
5	"leftBackSeat"
2	"rightBackSeat"

Fig. 9 Behavior exploration query: which places inside cars are preferred by the customer?

9 Evaluation

The Development Tool for Explorable XR Environments has allowed for the evaluation of the approach in terms of processing performance, the size of behavior representation and logs as well as the computational complexity of the log generation algorithm.

9.1 Performance

The performance of the tool has been evaluated in terms of transforming XR environments, inserting statements derived from annotations to a triplestore, and the number of FPS rendered for the generated explorable environments. The tests have been completed using:

1. *Work station 1* equipped with CPU Intel Core i7-5820K CPU 3.30GHz with 6 cores and 12 threads; 16 GB RAM 2400 MHz; GPU NVIDIA GeForce GTX 960; and HD Western Digital WD2003FZEX with 64 MB cache and rotational speed 7200 as well as the Windows 10 operating system.
2. *Work station 2* equipped with CPU Intel Core i7-4710HQ 2.50GHz with 4 cores and 8 threads; 8 GB RAM DDR3; GPU NVIDIA GeForce GTX 860M; and HD Hitachi TravelStar 7K1000 750 GB with 32MB cache and rotational speed 7200 as well as the Windows 10 operating system.

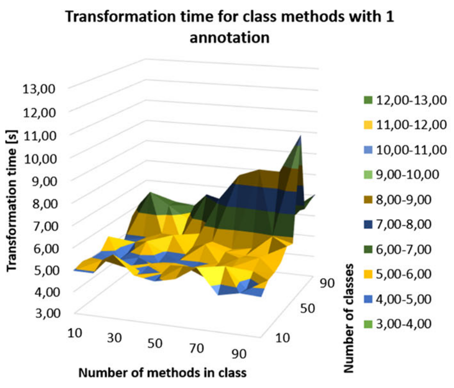
9.1.1 Transformation of XR environments

We have measured the time of transforming environments to their explorable counterparts using works station 1. We have generated environments with the number of classes in the range 10 to 100, step 10, every class including 10 to 100 methods with step 10. Each method has been assigned 1 to 5 annotations. The results of the transformation are presented in the graphs in Fig. 10. For every number of classes and every number of methods, 20 environments have been generated. Hence, every point in graphs a-e is the average time of 20 transformations. Graph f presents the accumulated time for all the environments including methods with 5 annotations.

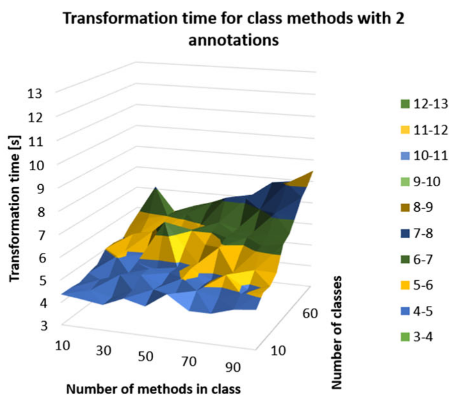
The results show that the transformation time growths linearly with the growth in the overall number of methods/annotations that are processed. This is confirmed by the high R-squared equal to 0.9803. However, the results presented in graphs a-e show that the influence of the number of annotations on the transformation time is moderate. The five-fold increase in the number of annotations leads to a slight increase in the average transformation time equal to 6.14/5.62=1.09. It means that the other activities encompassed by the transformation, such as the creation of directories and copying files of the project, are more time-consuming than processing annotations. This is also visible in graphs a-e, where the increase in the number of classes more affects the transformation time than the increase in the number of methods in a class.

9.1.2 Processing annotations

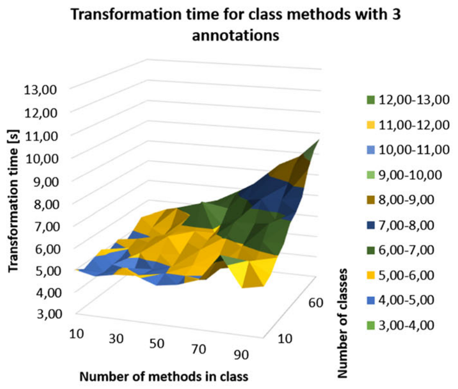
We have performed tests of streaming statements generated upon annotations at work station 2 to the triplestore installed on work station 1. The streams were generated by an environment with an animation of a moving object that lasted for 10 minutes. In every update of the object, statements from annotations about its position were added to the triplestore. The tests have been performed for 3 cases in terms of the location of the store: at



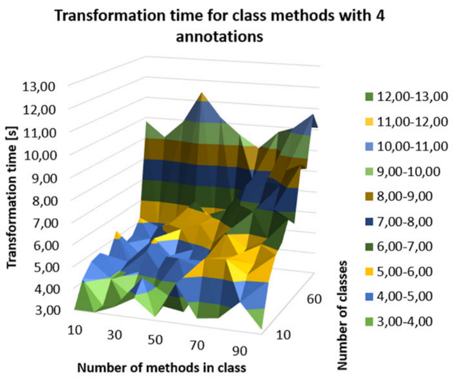
a)



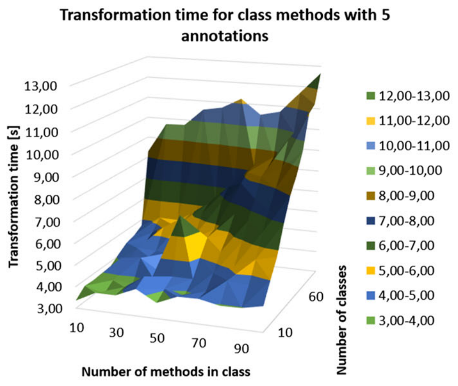
b)



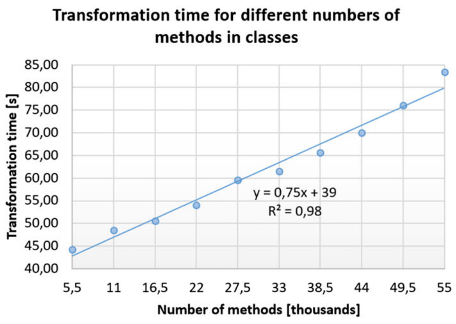
c)



d)



e)



f)

Fig. 10 Time of transforming XR environments including class methods with 1-5 annotation to explorable environments (**a**-**e**). Time of transformation for methods with 5 annotations depending on the overall number of methods (**f**)

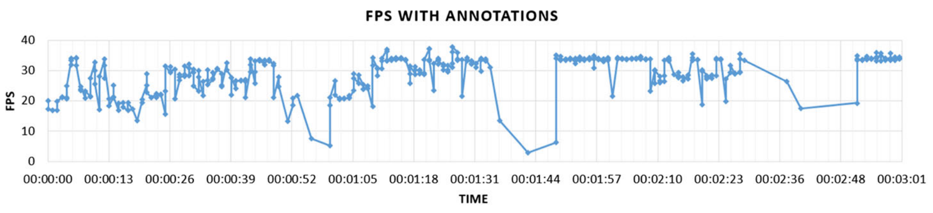
Table 2 Performance of adding statements derived from annotations to a triplestore as well as the size of the created annotations and generated logs

	property	localhost	LAN	internet
time [ms]	avg	27	209	2059
	stddev	3	56	78
	CV	0,12	0,27	0,04
	min	22	119	1786
	max	60	818	2188
number of	inserted annotations	20272	2732	288
	inserted statements	50680	6830	720
	inserted objects	15204	2049	216
	log [KB]	5108	655	85
size of	annotation [B]	258	246	303

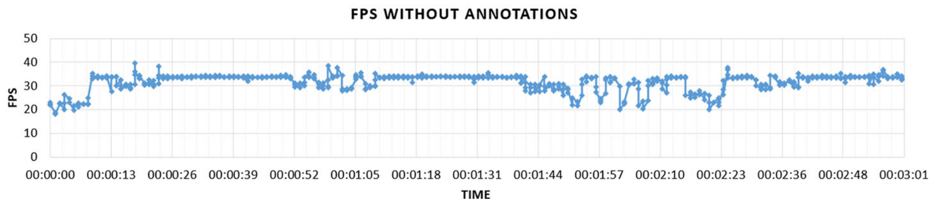
the localhost (together with the environment), on another host in the same LAN, and on another host on the internet. For every case, we repeated streaming 20 times. The results are summarized in Table 2. Whereas inserting statements to the store at the localhost required in average 27 ms, the performance for the LAN and the internet was much lower—209 ms and 2059 ms. It denotes significant network delay—182 and 2032 ms, respectively. Also, the number of FPS was lower in these cases as rendering and logging were executed by the same thread. The standard deviation of the results was relatively high for the LAN, which is expressed by the coefficient of variation (CV) equal to 0.27. CV was much lower for the localhost (0.12) and the internet (0.04), which denotes more stable transmissions.

9.1.3 FPS

We have used work station 2 to complete two sessions of using the immersive car showroom with the triplestore installed on the same station. The first session was completed using the explorable showroom, whereas the second session—using the prototype (non-explorable) showroom. For every session, we were registering the average number of FPS calculated for the last 10 frames. The sessions lasted for 3 minutes each. The results are shown in Fig. 11. While the average number of FPS for the session without annotations is 9% higher than with annotations, logging the behavior increases the coefficient of variation that is twice higher than without logging (0.2 to 0.1), which is also visible as the vertical distribution of the points in the graphs. The sessions consisted of three parts in which three cars were being watched from outside and inside and painted. Painting the cars was being executed continuously by the mouse move event occurring over a palette of colors. Therefore, it was generating streams with large numbers of statements about colors sent to the triplestore. This mostly decreased the number of FPS, which can be seen as the breakdowns in Fig. 11a in the intervals around 1:00, 1:42 and 2:40. In other periods of the session, 3D content was smoothly rendered with no visible difference in comparison to the session without logging.



a



b

Fig. 11 FPS for sessions of using the immersive car showroom: the explorable environment—with logging the behavior (**a**), and the prototype environment—without logging (**b**)

9.2 Size

We have evaluated the size of data used in the consecutive steps of the proposed method and generated while running explorable environments:

1) annotations created in Step 1 and the instructions generated on the basis of the annotations in Step 2,
2) classes and methods before and after the transformation in Step 2,
3) behavior logs generated while using explorable environments.

9.2.1 Annotations and generated instructions

We have compared the size of annotations created using the annotation module with the size of imperative instructions for logging behavior that were added to the classes and methods by the annotation compiler. This allowed investigating the gain in the developer's effort when using our approach in comparison to implementing the logging functions from scratch.

Using at least one annotation for a method in a class is followed by extending the class with additional auxiliary methods of the size equal to 911 bytes. The size of the annotation template (without parameters) is equal to 28 bytes, while the size of the generated instructions is equal to 729 bytes, which gives the gain ratio equal to 26.

9.2.2 Classes and methods

We have compared the overall size of classes and methods of the generated explorable environments to the size of classes and methods of their prototypes. The ratio also expresses the gain in the developer's effort when using our approach in comparison to implementing logging functions from scratch. For the comparison, we have used environments mentioned

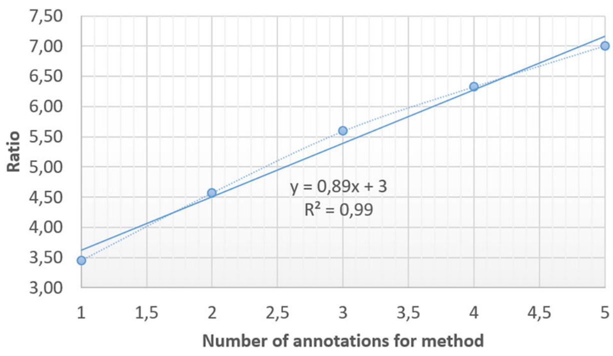


Fig. 12 The ratio of the size of generated methods/classes to the size of their annotated prototypes

in Section 9.1.1 with 100 classes, every one with 100 methods. All the methods have been assigned the number of annotations in the range 1 to 5. The results are presented in Fig. 12. The gain ratio is proportional to the number of annotations for a method and varies from 3.5 for methods with a single annotation to 7 for methods with 5 annotations.

9.2.3 Behavior logs

The size of the behavior logs containing the streamed statements mentioned in Section 9.1.2 is summarized in Table 2. The largest log (5108 KB) was generated in the case of using the XR environment with the triplestore at the same work station. It is almost 8 times larger than the log collected remotely over the LAN, and 60 times larger than the log collected over the internet. The proportions are similar for the numbers of annotations represented within the logs, statements derived from the annotations, and objects used in the statements. In the cases, the average size of the statements derived from an annotation is similar, ranging from 258 to 303 bytes.

The size of the behavior log representing the three minutes long session of using the explorable car showroom, mentioned in Section 9.1.3, equals to 98 KB. It consists of 990 statements on 279 objects derived from 273 annotations.

9.3 Computational complexity

The computational complexity of the log generation algorithm described in Section 6 is the following for the particular steps of the algorithm.

1. Determining the values of the `subject`, `predicate` and `object` requires 3 operations, thus the complexity is *O(1)*.
2. Creating and inserting a statement derived from an annotation to a triplestore requires 2 operations, thus the complexity is *O(1)*.
3. Determining the value of a `timestamp` requires 1 operation, thus the complexity is *O(1)*.
4. The complexity of the substeps completed for an annotation is as follows.
   (a) Finding the statement with the latest interval in a list of statements sorted by the intervals requires 1 operation, this is an operation *O(1)*.

(b) The complexity of the substeps completed for a `timestamp` is as follows.

   i The creation of a `time point` requires 1 operation, thus the complexity is *O(1)*.
   ii The creation of a `time interval` requires 1 operation, thus the complexity is *O(1)*.
   iii The creation of a `subject time slice` requires 1 operation, thus the complexity is *O(1)*.
   iv Assigning the created `time point` or `interval` to the subject requires 1 operation, thus the complexity is *O(1)*.
   v Setting the `predicate` value for the `subject` requires 1 operation, thus the complexity is *O(1)*.
   vi The complexity of the substeps completed for a `predicate` is as follows.

      A The creation of an `object time slice` requires 1 operation, thus the complexity is *O(1)*.
      B Setting the `predicate` value for the `subject` requires 1 operation, thus the complexity is *O(1)*.
      C Assigning the created `time point` or `interval` to the object requires 1 operation, thus the complexity is *O(1)*.

Thereby, the overall computational complexity of processing 1 annotation by the algorithm is *O(1)*, and *O(n)* for processing n annotations.

10 Discussion

The proposed approach offers a qualitative contribution over the available methods and tools for development of XR environments, by satisfying the requirements listed in Section 3.

10.1 Evaluation results

The obtained results show that the implemented environment efficiently transforms XR environments to their explorable counterparts, as the time of transformation varies from 3 seconds for smaller environments to 13 seconds for larger environments. Such a reasonable transformation time allows for developing web services transforming XR environments on-demand.

The time of inserting statements derived from annotations strongly depends on the relative location of the explorable environment and the target triplestore. Its lowest value obtained for the environment and triplestore installed on the same host enables logging semantic information about animations, maintaining an acceptable number of FPS. It is otherwise in the case of transmitting the statements over the LAN or the internet when the delay affects the quality of the visualization. This problem may be solved in different manners. First, logging behavior can be performed by a different thread than the one responsible for updating and rendering 3D content. In such a case, the statements can be waiting in a queue to be saved to the triplestore, without stopping the primary graphical thread. Second, the transmission can gather multiple statements derived from various annotations into larger packages to reduce the average delay. Third, other triplestores should be tested in terms of efficient communication over a network. Currently, the network delay is no problem for logging actions and interactions that do not generate streams of statements to be registered (like in animations).

The high performance of the log generation algorithm itself has been confirmed by the theoretical analysis of its computational complexity, which linearly depends on the number of annotations to be processed.

The behavior logs generated in the analyzed use cases are of acceptable size taking into account possible practical applications of the approach, such as marketing, merchandising, and training sessions, even in cases of streaming annotations to a triplestore. The comparison of the size of annotation to the size of the generated instructions as well as the comparison of the size of classes and methods before and after the transformation show the large gain in using the approach rather than implementing the corresponding functions from scratch using programming languages and libraries.

10.2 Guidelines for developers

Processing annotations by the algorithm inserts new code instructions at the end of the class methods annotated in Step 1 (cf. Section 5.1) of the proposed method. This requires that the variables used in annotations be accessible at the end of the methods in the form in which they should be registered. Therefore, we can specify the following guidelines for developers who use our approach to create new XR environments that can potentially be made explorable or to adjust existing environments before their transformation.

1. Local method variables that will be referred from within annotations should be defined in the main scope of the method but not in narrower scopes, e.g., within conditional and loop instructions.
2. A method that will be annotated should finalize with the values of the local and global variables as well as method parameters to be annotated equal to the values used in the method. Setting appropriate values should be done at the beginning of the method.

Although the guidelines introduce certain restrictions on the way in which the approach is used, they may be applied by developers before transforming an environment by minor changes of the code if needed. The following modifications should be accomplished if the aforementioned conditions are not met.

1. If an annotation refers to a local method variable that is not defined in the main scope of the method, the variable declaration should be moved to the main scope.
2. If a method, after completing its main job, is preparing new values of variables to be used by other methods later on, the preparation should be moved to those other methods while leaving the variables unchanged at the end of the primary method.
3. If a method, after completing its main job, is invoking other methods:
    (a) Copies of the variables to be logged should be passed to the methods instead of references to the variables. This is required to prevent modification of the variables by the other method invoked, which would lead to logging invalid values, or
    (b) The invocations should be moved before the method does its main job.

    In particular, it is relevant to methods with tail recursion, which might be required to be exchanged with head recursion or a loop.
4. If it is necessary to register actions and interactions executed by methods implemented in external libraries that are available only as binaries, the methods should be wrapped

by additional annotated methods in the environment. For instance, if an environment invokes a web service to get a new representation of an object in a scene, which should be registered, the invocation should be moved to an extra method, which will be annotated, thus logging the execution of the service.

11 Conclusions and future work

In this paper, we have presented the approach to creating explorable XR environments, in which actions and interactions of objects are registered in a way that permits their further temporal exploration with queries. The behavior logs generated using the algorithm we propose conform to the temporal representation of 3D content behavior. The representation is based on the semantic web standards. It allows for expressing domain knowledge, and for inference of tacit knowledge. The logs are finally subject to semantic queries about users' and objects' actions and interactions.

Due to the focus on annotating class methods, which often have domain-related names and parameters, and the possibility of using knowledge in any domain, the approach can encourage average users and domain experts with limited programming skills to contribute to the development of XR. In addition, with the approach, explorable environments do not have to be created from scratch but may be generated from available systems. These advantages set directions to a variety of new applications of XR in the diversity of domains.

Future development encompasses several aspects. First, we plan to extend our tool with the possibility of the presentation of past actions and interactions that would accompany query-based knowledge acquisition. It can be achieved in two ways. On the one hand, the animated 3D content of the environment can be recorded as a movie, which can be further replayed from specific points in time to illustrate desirable content behavior. On the other hand, dumps of the application memory can be created and restored in response to queries to present actions and interactions logged in a session of using the environment. Such dumps would need to cover the states of the objects that can potentially be subject to exploration. Second, a tool for graphical modeling of annotations in the form of graphs associating methods, classes, and packages of code can be implemented. Third, the implementation of network communication in the development tool needs to be improved to enable logging semantic information about animations generating streams of statements to a triplestore. We plan to evaluate other existing libraries and triplestores and consider our own implementation of critical components. Finally, the implementation could be further improved in case of the availability of aspect-oriented libraries compatible with game engines.

Acknowledgements The author would like to thank prof. Krzysztof Walczak in the Department of Information Technology at the Poznań University of Economics and Business (PUEB) for his valuable remarks on this work. The author would also like to thank Adrian Nowak—a developer at PUEB, for the implementation of the immersive car showroom used as an illustrative example in this paper.

References

1. Autodesk (2020a) 3ds Max. https://www.autodesk.pl/products/3ds-max/overview
2. Autodesk (2020b) Motion builder. https://www.autodesk.com/products/motionbuilder/overview
3. Ben Ellefi M, Drap P, Papini O, Merad D, Royer J, Nawaf M, Nocerino E, Hyttinen K, Sourisseau J, Gambin T, et al. (2019) Ontology-based web tools for retrieving photogrammetric cultural heritage models. Underwater 3D Recording & Modeling ISPRS, Limassol
4. Berners-Lee T, Hendler J, Lassila O (2001) The semantic web. Sci Am 284(5):34–43. http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21
5. Blender Foundation (2020) Blender. http://www.blender.org
6. Brickley D, Miller L (2014) FOAF vocabulary specification 0.99. http://xmlns.com/foaf/spec/
7. Chmielewski J (2014) Finding interactive 3D objects by their interaction properties. Multimedia Tools and Applications 69(3):773–798. https://doi.org/10.1007/s11042-012-1125-x
8. Chu Y-L, Li T-Y (2012) Realizing semantic virtual environments with ontology and pluggable procedures. Applications of Virtual Reality, 171. BoD–Books on Demand
9. DCMI (2020) Dublin core metadata initiative. https://www.dublincore.org/specifications/dublin-core/dcmi-terms/
10. De Troyer O, Kleinermann F, Mansouri H, Pellens B, Bille W, Fomenko V (2007a) Developing semantic VR-shops for e-Commerce. VR 11(2):89–106. https://doi.org/10.1007/s10055-006-0058-y
11. De Troyer O, Kleinermann F, Pellens B, Bille W (2007b) Conceptual modeling for virtual reality. In: Grundy J, Hartmann S, Laender AHF, Maciaszek L, Roddick JF (eds) Tutorials, posters, panels and industrial contributions at the 26th Int. Conference on Conceptual Modeling - ER 2007, Auckland, New Zealand, CRPIT, vol 83, pp 3–18
12. Divakaran A (2001) An overview of MPEG-7 motion descriptors and their applications. In: Skarbek W (ed) Computer analysis of images and patterns. Springer, Berlin, pp 29–40
13. Drap P, Papini O, Sourisseau JC, Gambin T (2017) Ontology-based photogrammetric survey in underwater archaeology. In: European semantic web conference, Springer pp 3–6
14. Facebook Technologies (2018) Oculus rift. https://support.oculus.com
15. Flotyński J, Nowak A (2019) Annotation-based development of explorable immersive VR/AR environments. In: International conference on 3d immersion (IC3d), December 11-12, Brussels, Belgium
16. Flotyński J, Walczak K (2013) Describing semantics of 3D web content with RDFa. In: The first international conference on building and exploring web based environments, Sevilla (Spain), January 27 - February 1, 2013, ThinkMind, pp 63–68
17. Flotyński J, Walczak K (2013) Semantic Modelling of Interactive 3D Content. In: Proceedings of the 5th joint virtual reality conference, Paris, France
18. Flotyński J, Walczak K (2014) Multi-platform semantic representation of interactive 3D content. In: Proceedings of the 5th doctoral conference on computing, electrical and industrial systems, April 7-9, Lisbon, Portugal
19. Flotyński J, Walczak K (2016) Customization of 3D content with semantic meta-scenes. Graph Model 88:23–39. https://doi.org/10.1016/j.gmod.2016.07.001
20. Flotyński J, Walczak K (2017) Ontology-based representation and modelling of synthetic 3D content: A state-of-the-art review. Comput Graph Forum 35:329–353. https://doi.org/10.1111/cgf.13083
21. Flotyński J, Nowak A, Walczak K (2018) Explorable representation of interaction in VR/AR environments. In: Proceedings of AVR proceedings of AVR 2018, lecture notes in computer science, Springer International Publishing, pp 589–609
22. Flotyński J, Brutzman D, Hamza-Lup FG, Malamos A, Polys N, Sikos LF, Walczak K (2019) The semantic web3d: towards comprehensive representation of 3D content on the semantic web. In: International conference on 3d immersion (IC3d), December 11-12, 2019, Brussels, Belgium
23. Games E (2020) Unreal engine. https://www.unrealengine.com/
24. García-Rojas A, Vexo F, Thalmann D, Raouzaiou A, Karpouzis K, Kollias S (2006) Emotional body expression parameters in virtual human ontology. In: Proceedings of 1st international workshop on shapes and semantics, Matsushima, Japan, June 2006, pp 63–70
25. Gruber T (2009) Encyclopedia of database systems. http://tomgruber.org/writing/ontology-definition-2007.htm
26. Gutiérrez M, García-Rojas A, Thalmann D, Vexo F, Moccozet L, Magnenat-Thalmann N, Mortara M, Spagnuolo M (2007) An ontology of virtual humans: Incorporating semantics into human shapes. Visual Computer 23(3):207–218
27. Gownder JP, Voce C, Mai M, Lynch D (2016) Breakout vendors: virtual and augmented reality. https://www.forrester.com/report/Breakout+Vendors+Virtual+And+Augmented+Reality/-/E-RES134187/

28. ISO (2015) ISO/IEC 15938-13:2015 [ISO/IEC 15938-13:2015]. Information technology — multimedia content description interface - Part 13: Compact descriptors for visual search. https://www.iso.org/standard/65393.html
29. Kalogerakis E, Christodoulakis S, Moumoutzis N (2006) Coupling ontologies with graphics content for knowledge driven visualization. In: VR '06 Proceedings of the IEEE conference on virtual reality, Alexandria, Virginia, USA, pp 43–50
30. Krötzsch M, Simancik F, Horrocks I (2012) A description logic primer. arXiv:1201.4089
31. Leap Motion (2018) Leap motion documentation. https://developer.leapmotion.com/documentation
32. Leipzig University UniversityofMannheim (2019) DBpedia. https://wiki.dbpedia.org/
33. Lugrin JL (2009) Alternative reality and causality in virtual environments. PhD thesis University of Teesside, Middlesbrough, United Kingdom
34. Milgram P, Takemura H, Utsumi A, Kishino F (1995) Augmented reality: a class of displays on the reality-virtuality continuum. In: Telemanipulator and telepresence technologies, international society for optics and photonics, vol 2351, pp 282–292
35. Nowak A, Flotyński J (2018) A virtual car showroom. In: Proceedings of the 23rd international ACM conference on 3D web technology, association for computing machinery, New York, NY, USA, Web3D'18. https://doi.org/10.1145/3208806.3208832
36. Parr T (2014) ANTLR. https://www.antlr.org/
37. Pellens B, Kleinermann F, De Troyer O (2009) A development environment using behavior patterns to facilitate building 3D/VR applications. In: Proceedings of the 6th australasian conference on international entertainment, ACM, IE '09, pp 8:1–8:8
38. Perez-Gallardo Y, Cuadrado JLL, Crespo ÁG, de Jesús CG (2017) GEODIM: A semantic model-based system for 3D recognition of industrial scenes. In: Current trends on knowledge-based systems, Springer, pp 137–159
39. Poznań University of Economics and Business (2018) A virtual car showroom. https://www.youtube.com/watch?v=qdM10ErmsXQ
40. Rabattu PY, Massé B, Ulliana F, Rousset MC, Rohmer D, Léon JC, Palombi O (2015) My Corporis Fabrica Embryo: An ontology-based 3D spatio-temporal modeling of human embryo development, vol 6. https://doi.org/10.1186/s13326-015-0034-0
41. Semiodesk GmbH (2015a) Semiodesk TinyVirtuoso. https://bitbucket.org/semiodesk/tinyvirtuoso/
42. Semiodesk GmbH (2015b) Semiodesk trinity. https://bitbucket.org/semiodesk/trinity/
43. Sikos LF (2017a) 3D Model indexing in videos for content-based retrieval via X3D-based semantic enrichment and automated reasoning. In: Proceedings of the 22Nd international conference on 3D web technology, ACM, New York, NY, USA, Web3D '17, pp 19:1–19:7
44. Sikos LF (2017b) Description logics in multimedia reasoning. Springer International Publishing, https://doi.org/10.1007/978-3-319-54066-5
45. Technologies U (2020) Unity. http://unity.com/
46. Trellet M, Ferey N, Baaden M, Bourdot P (2016) Interactive visual analytics of molecular data in immersive environments via a semantic definition of the content and the context. In: Immersive analytics (IA), 2016 Workshop on, IEEE pp 48–53
47. Trellet M, Férey N, Flotyński J, Baaden M, Bourdot P (2018) Semantics for an integrative and immersive pipeline combining visualization and analysis of molecular data. J Integr Bioinform 15(2):1–19
48. Vasilakis G, García-Rojas A, Papaleo L, Catalano CE, Robbiano F, Spagnuolo M, Vavalis M, Pitikakis M (2010) Knowledge-Based Representation of 3D media. Int J Softw Eng Knowl Eng 20(5):739–760
49. Walczak K, Flotyński J (2015) Semantic Query-based Generation of Customized 3D Scenes. In: Proceedings of the 20th international conference on 3D web technology, ACM, New York, NY, USA, Web3D '15, pp 123–131, https://doi.org/10.1145/2775292.2775311
50. Walczak K, Flotyński J (2015) Ontology-based creation of 3D content in a service-oriented environment. In: Abramowicz W (ed) Lecture notes in business information processing: business information systems, vol 208. Springer, Heidelberg, pp 77–89, https://doi.org/10.1007/978-3-319-19027-3. ISBN 978-3-319-19026-6
51. Web3D Consortium (2019) X3D ontology for semantic web. https://www.web3d.org/x3d/content/semantics/semantics.html
52. Welty C, Fikes R (2006) A reusable ontology for fluents in OWL. In: Proceedings of the 2006 conference on formal ontology in information systems: proceedings of the fourth international conference (FOIS 2006), IOS Press, Amsterdam, The Netherlands, pp 226–236. http://dl.acm.org/citation.cfm?id=1566079.1566106
53. Wu Y, He F, Zhang D, Li X (2018) Service-oriented feature-based data exchange for cloud-based design and manufacturing. IEEE Trans Serv Comput 11(2):341–353
54. W3C (2012) OWL. https://www.w3.org/TR/owl2-syntax/

55. W3C (2012) OWL 2 web ontology language profiles (Second Edition). http://www.w3.org/TR/owl2-profiles/#Computational_Properties
56. W3C (2013) SPARQL. https://www.w3.org/TR/sparql11-query/
57. W3C (2014a) RDF. https://www.w3.org/TR/rdf11-concepts/
58. W3C (2014b) RDFS. https://www.w3.org/TR/rdf-schema/
59. W3C (2017) X3D. http://www.web3d.org/getting-started-x3d
60. Zhang D, He F, Han S, Li X (2016) Quantitative optimization of interoperability during feature-based data exchange. Integr Comput-Aid Eng 23(1):31–50