Visual aspect-oriented modeling of explorable extended reality environments

Jakub Flotyński¹

Received: 31 March 2020 / Accepted: 21 October 2021 / Published online: 27 November 2021
© The Author(s) 2021

Abstract

The availability of various extended reality (XR) systems for tracking users' and objects' behavior opens new opportunities for analyzing users' and objects' interactions and autonomous actions. Such analysis can be especially useful and attainable to domain experts when it is based on domain knowledge related to a particular application, liberating the analysts from going into technical details of 3D content. Analysis of XR users' and objects' behavior can provide knowledge about the users' experience, interests and preferences, as well as objects' features, which may be valuable in various domains, e.g., training, design and marketing. However, the available methods and tools for building XR focus on 3D modeling and programming rather than knowledge representation, making them unsuitable for domain-oriented analysis. In this paper, a new visual approach to modeling explorable XR environments is proposed. It is based on a semantic representation of aspects, which extend the primary code of XR environments to register their behavior in a form explorable with reasoning and queries, appropriate for high-level analysis in arbitrary domains. It permits domain experts to comprehend and analyze what happened in an XR environment regarding users' and objects' actions and interactions. The approach has been implemented as an extension to MS Visual Studio and demonstrated in an explorable immersive service guide for household appliances. The evaluation results show that the approach enables efficient development of explorable XR and may be useful for people with limited technical skills.

Keywords Extended reality · 3D web · Exploration · Semantic web · Ontologies · Aspect-oriented modeling

1 Introduction

Extended reality (XR) covers different forms of combined real and virtual environments, which range from augmented reality (AR) to virtual reality (VR) in the reality–virtuality continuum (Milgram et al. 1995), encompassing different types of presentation and interaction with objects (Gownder et al. 2016). XR becomes widely used in multiple application domains, such as training, design, marketing, merchandising, education and engineering, due to the growing range of available devices with increasing performance and falling prices. Users and 3D objects in XR environments can behave, which includes autonomous actions (e.g., a flying aircraft) and interactions (e.g., a serviceman repairing a device). Such behavior is typically expressed by animations

leading to the creation, modification and destruction of 3D objects in scenes.

XR offers a high potential for tracking users' and objects' behavior using systems for motion and gesture capture, eye tracking as well as trackable headsets and interactive controllers. Multiple application domains can benefit from registering the behavior of XR users and 3D objects, including their actions and interactions, in an explorable way. The analysis of behavior can be especially useful when collected information is represented using knowledge in a particular domain, comprehensible to experts in the field. Furthermore, registered behavior can be subject to reasoning, which permits to infer implicit (tacit) knowledge from the registered knowledge, as well as queries about environment states at different moments and periods in time. The acquired knowledge can serve to comprehend 3D objects' behavior and users' behavior, experience, interests and preferences.

Examples of exploration of users' and objects' behavior can be given for diverse application domains. For example, information collected while training can be used to consider

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

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

diverse situations and teach beginners. Information collected during a design process can enable analysis of the project at its different stages, including particular designers' contributions and the consistency with the original requirements. Collected information about customers' activities can help discover their interests and preferences for marketing and merchandising purposes. In turn, it may facilitate arrangement of real stores as well as preparation of personalized offers. Collected information about the states and behavior of appliances can serve to analyze how they work and to identify possible faults. In tourism, information about virtual guided tours can be used to join the most interesting ones and address customers' interests in the program of the tours. In medicine, collected information about the steps of virtual treatments can be analyzed to teach students.

However, the available technologies, including 3D formats, programming languages, 3D modeling environments and game engines, have not been intended for the goals mentioned above. These solutions have been designed for 3D representation, modeling and programming but not for knowledge representation and modeling, which is essential to domain-oriented analysis of users' and objects' behavior. Hence, the available approaches are intelligible to graphics designers and programmers rather than average users and domain experts, who use XR for different application domains but typically do not have advanced technical skills.

The main contribution of this paper is an approach to *visual aspect-oriented modeling of explorable XR environments*. The concept of the approach is depicted in Fig. 1. An **explorable XR environment** is an XR environment in which users' and 3D objects' behavior, including actions and interactions, is logged and represented using domain knowledge combined with visual descriptors. The approach enables creation of explorable XR environments and transformation of existing environments to their explorable counterparts. The behavior representation is based on the semantic web approach, which enables reasoning and queries about environment states at different moments and intervals in time, stored in the form of *visual semantic behavior logs*. The results of queries encompass knowledge and visualization of what happened in an environment, including the creation, modification and destruction of 3D objects in the scene. The

approach is built upon imperative programming languages (e.g., C#, C++, Java and JavaScript) and game engines (e.g., Unity) using aspect-oriented programming. The aspect-oriented approach minimizes the effort and modifications of code necessary to transform available XR environments to their explorable equivalents. Finally, the visual combination of the semantic web and aspect-oriented programming can disseminate the creation and use of XR among people with limited programming skills.

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. In Sect. 3, the ontology-based behavior representation, which is used to generate behavior logs, is presented. Section 4 presents the logger ontology, which is used to describe how XR behavior is logged. Section 5 presents the pipeline of modeling explorable XR environments using the logger ontology. The approach is presented along with the Visual Logger Editor, which implements the pipeline in MS Visual Studio. Further, the approach is illustrated with an explorable immersive service guide for home appliances in Sect. 6. It is followed by the evaluation (Sect. 7) and discussion (Sect. 8). Finally, Sect. 9 concludes the paper and indicates possible future research.

2 Related works

2.1 Modeling 3D content behavior

3D content behavior covers actions performed by 3D objects and users as well as interactions between objects and 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 actions and interactions of 3D content, including 3D modeling tools [e.g., 3ds Max (Autodesk 2020a) and Blender (Blender Foundation 2020)], animation modeling tools [e.g., Motion Builder (Autodesk 2020b)] as well as game engines [e.g., Unity (Technologies 2020) and Unreal (Games 2020)]. 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 that do not require programming skills use state diagrams and keyframes to interpolate objects' properties, e.g., geometry, position, orientation and colors.

The tools use multiple 3D content formats, e.g., FBX, OBJ and VRML. However, since the formats have been intended strictly for 3D content, they neither represent domain-specific semantics of 3D content nor content exploration with reasoning and queries. Even more complex formats, such as the Extensible 3D (X3D) (W3C 2017), which

Fig. 1 The concept of visual aspect-oriented modeling and use of explorable XR environments

enable encoding of metadata, lack expressive formalism for content description, such as hierarchies of classes and properties as well as relationships between them. The description of domain-specific semantics is beyond the scope of the available 3D content formats and requires additional tools.

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 (W3C 2012b). The semantic web is based on description logics, which are a well-recognized field comprehensively described in multiple publications, e.g., Baader et al. (2010, 2017). Content descriptions based on the semantic web are human-readable and computer-processable. Therefore, the semantic web has been chosen to be the foundation of the approach proposed in this paper.

The main standards used to represent content of any type on the semantic web are the resource description framework (RDF) (W3C 2014a), the RDF Schema (RDFS) (W3C 2014b) and the Web Ontology Language (OWL) (W3C 2012a). RDF is a data model based on statements in the form of triples (*subject*, *predicate*, *object*). In a triple, the *subject* is what is described, the *predicate* is a property of the subject, and the *object* is a predicate value, a descriptor, or another entity that is in the relationship with the subject. An example of an RDF triple is (*user*, *turns on*, *appliance*). A graphical representation of the triple in the Protégé OntoGraf editor (Stanford University 2020) is shown in Fig. 2.

RDFS and OWL are languages built upon RDF, providing higher expressiveness—classes and properties with relationships and hierarchies, which enable content description. Properties are used as predicates in RDF triples. RDFS specifies *domains* and *ranges* for properties. A property domain is a class whose objects can be described using the property. A property range is a class whose objects or literal values (e.g., string, integer and float) can be indicated by the property. OWL specifies object properties, whose range is classes, and datatype properties, whose range is literal values.

Fig. 2 Example of graphical RDF triple representation

These standards permit design of *ontologies*, which are specifications of a formal conceptualization for a domain (Gruber 2009) or common sense human knowledge, i.e., an abstract, simplified view of a fragment of the world (Sikos 2017b). Ontologies enable knowledge representation, which can be expressed using statements that belong to two groups. *TBox* is a set of terminological knowledge statements, which describe a conceptualization—a set of concepts—and properties for these concepts. *ABox* is a set of assertional knowledge statements, which describe facts about individuals (objects) using concepts specified in a TBox (Sikos 2017b). Ontologies developed with RDF, RDFS and OWL can be queried using SPARQL (W3C 2013), which is the main query language on the semantic web.

Ontologies constitute the foundation of the semantic web, which is suitable for arbitrary applications and domains. The semantic web is currently one of the leading approaches to knowledge representation and one of the main trends in the evolution of the web (Berners-Lee et al. 2001). It gains increasing attention in the context of XR, e.g., for photogrammetry (Ben Ellefi et al. 2019), molecular visualization (Trellet et al. 2016, 2018), content description and retrieval (Sikos 2017a, b), design of industrial spaces (Perez-Gallardo et al. 2017) and archeology (Drap et al. 2017). Finally, ontologies are a more strict, structured and comprehensive tool for content description than typical metadata based on schemes with properties and keywords. The semantic web approach is independent of particular application domains. Therefore, its usage in a domain (e.g., XR) typically requires creating specific ontologies.

The expressiveness of an ontology is determined by the OWL profile used to encode the ontology (W3C 2012b). OWL profiles implement different sets of description logics constructors, such as intersection, union, complement as well as existential, universal, and cardinality restriction. The constructors can be compared to production and decision rules, which have the form of *if-then* clauses. On the one hand, as opposed to production and decision rules, the constructors do not enable expression of arbitrarily complex *if-then* clauses, which limits the expressiveness of the ontologies that can be created. On the other hand, it makes the reasoning problems for the possible ontologies (such as instance checking, concept satisfiability, concept subsumption, ontology consistency and query answering) known in terms of decidability and computational complexity. It is an essential advantage over the other approaches in practical application domains.

Ontologies can also be compared to other approaches in terms of processing complex queries, which combine multiple conditions. Query processing is an inherent part of databases (e.g., relational and object-oriented) as well as semi-structured datasets (e.g., XML and JSON documents) in various systems. The largest advantage of ontologies over

the other solutions is the formal semantics specified by the semantic web standards, which enable reasoning. Hence, unlike the other approaches, ontologies permit queries about both explicit information and implicit information, which can be automatically inferred. It enables content authors to specify only fundamental (non-inferable) information, liberating them from explicitly specifying all information that could be subject to potential queries. Further, it reduces the overall size of the datasets and helps preserve data consistency.

2.3 Metadata and semantics for 3D behavior

Several solutions have been devised to describe metadata and semantics of 3D behavior. The analyzed solutions are summarized in Fig. 3.

The MPEG-7 standard (ISO 2015) 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 Divakaran (2001). The descriptors cover motion activities and trajectories, camera motion and warping. In addition, the standard specifies the XML-based Description Definition Language, which specifies the syntax for creating new descriptors and description schemes.

In Chmielewski (2008, 2012a, b), an approach to describing metadata for interaction of 3D objects has been presented. The interaction model encompasses events, conditions and actions encoded using XSD schemes. In addition, a query language about interaction metadata, with syntax similar to SQL, has been proposed. In the paper, an example related to the description of virtual heritage objects interaction is presented.

Several approaches enable representation and modeling of 3D content behavior with ontologies and semantic web standards, which permit more complex conceptualization than typical metadata. An extensive state-of-the-art review of ontology-based modeling and representation of 3D content is presented in Flotyński and Walczak (2017)

and Flotyński (2020). In this paper, only the most specific solutions are outlined.

The approach proposed in Chu and Li (2008, 2012) uses ontologies to build multi-user virtual environments and avatars. The focus is on representing the geometry, space, animation and behavior of 3D content. The covered concepts are semantic equivalents of concepts 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 using descriptors.

In the approach proposed in Kleinermann et al. (2005), Pellens et al. (2005a, b, 2006, 2008, 2009) and De Troyer et al. (2007a, b), 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. The implemented behavior can be exported to X3D scenes.

The ontology proposed in Vasilakis et al. (2010) specifies classes and properties that represent animations using keyframes, which are linked to geometrical and structural descriptions of the objects.

The ontology of virtual humans (Gutiérrez et al. 2007; García-Rojas et al. 2006) consists of geometrical descriptors (for vertices and polygons), structural descriptors (for articulations levels), 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 Kalogerakis et al. (2006) 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, individuals of the atom class (body) are represented by spheres as parts of chemical compounds (head). Thus, final 3D content is based on both the explicit and implicit (inferred) knowledge.

In the simulation environment presented in Lugin (2009), the Unreal engine enables rigid-body physics and content presentation. In addition, an inference engine enables reasoning and updates the scene representation when events occur. A behavioral engine enables action recognition and changes of conceptual objects' properties. The use of different engines within one environment permits separation

	Type	Specificity level	Example	Flexible 3D-domain connection
Divakaran (2001)				
ISO (2015)	metadata	3D	video browsing	✓
Chmielewski (2008, 2012a, 2012b)	metadata	3D	virtual museum	-
Chu and Li (2008, 2012)	semantics	3D	virtual humans	-
De Troyer et al. (2005–2009)	semantics	3D and domain	game design	✓
Gutiérrez et al. (2007)				
García-Rojas et al. (2006)	semantics	domain	virtual humans	-
Kalogerakis et al. (2006)	semantics	3D	genome visualization	-
Lugin (2009)	metadata	domain	rigid body simulation	-
Rabattu et al. (2015)	semantics	domain	human embryo	✓
Trellet et al. (2016)	semantics	domain	molecular visualization	-
Flotyński et al. (2014–2019)	semantics	3D and domain	virtual museum	✓

Fig. 3 Metadata- and semantics-based representation of 3D behavior

of concerns between users with different expertise while developing XR.

An approach to spatio-temporal reasoning over ontology-based representations of evolving human embryos has been proposed in Rabattu et al. (2015). The ontologies are encoded in RDF and OWL, and they describe stages, periods and processes.

In Trellet et al. (2016), an ontology-based representation of 3D molecular models has been proposed. The approach combines different input (e.g., interaction with 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 Flotyński and Walczak (2014, 2015) and Walczak and Flotyński (2015) uses ontologies to represent 3D content at different specificity levels—related to 3D as well as an application domain. It also enables reasoning to liberate content authors from determining all content properties. Next, semantic queries to generic 3D templates (meta-scenes) can generate customized 3D scenes. Finally, the customized scenes are transformed to different 3D formats processable by 3D browsers.

In Flotyński et al. (2019a), the works of the X3D Semantic Web Working Group (Web3D Consortium 2020) have been outlined. In particular, an approach to generating ontology-based 3D formats from available 3D formats has been proposed. The X3D Ontology (Web3D Consortium 2019), which is a semantic equivalent of the Extensible 3D (X3D) (W3C 2017) 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.

So far, the semantic representation of behavior, including actions and interactions, has gained little attention compared to the representation of other, static, content features such as geometry, structure and appearance. However, a few works independently of XR content have addressed the semantic representation of mutable objects' properties.

2.4 Temporal ontology-based content representation

Semantic temporal representation of content has been extensively studied in the domain of the semantic web. The available solutions encompass: temporal description logics (Artale and Franconi 2001), temporal RDF (Gutierrez et al. 2005), versioning of ontologies (Klein and Fensel 2001) and *n*-ary relations (Noy and Rector 2006). The problems associated with the use of the approaches mentioned above have been

summarized in Welty and Fikes (2006) and Batsakis et al. (2009). They are mostly related to incompatibility with the semantic web standards as well as problems with automated reasoning.

Another solution, which avoids those shortcomings, are 4D-fluents (Welty and Fikes 2006). A *fluent* is a property that varies over time. Every statement can be transformed to express temporal information. For the statement *two objects are linked by a property*, the temporal equivalent is the statement *two objects are linked by the 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 points* or *time intervals*. Time points are used for instant statements, whereas time intervals are used for temporary statements. The representation of an object that has several different values of a property in different points or intervals 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. The following steps must be completed to add temporal information to the statement: *object1 is linked to object2 by property*.

1. For both *object1* and *object2*, create time slices *object1TS* and *object2TS*, and set the *isTimeSliceOf* property of both time slices to their primary objects.
2. Create a *time point* or *time interval* object, and set the *hasTimePoint* or *hasTimeInterval* property of the time slices to the *point/interval*.
3. Link the time slices by the *property* as the primary objects were linked.

For instance, to express that a virtual customer was watching a virtual car for 10 minutes, create time slices for both the customer and the car, and link them by the *watches* property. Next, create the interval representing those 10 minutes, and assign it to the time slices.

Although the presented temporal approaches are independent of XR, their potential application also covers this domain.

2.5 Aspect-oriented programming

The aspect-oriented programming paradigm complements object-oriented programming by enabling extension of application behavior with minimal modifications of the available code. It is possible due to the use of *aspects*, which cut across different software modules, classes and methods. Clear definitions related to aspect-oriented programming can be found in Spring Framework (2020). An *aspect* is a set of functions (referred to as *advice*s), which implement new behavior of an application that should be executed in

different places in the code, including different methods of different classes as well as different points within the methods. A place in the application code where an invocation of an advice is injected is called a *join point*. Code annotations and attributes are typically used to indicate advices and join points, for example, in the Spring library for Java (Spring Framework 2020) and the PostSharp library for C# (SharpCrafters 2020).

For example, an aspect may contain new functions that can be added to an application to log transactions. In such a case, advices are functions responsible for logging the source and destination accounts, their balance and the transaction status. In turn, join points are the beginnings, exits and throw-exception clauses in the relevant methods in the application code. In such an application, once a join point is reached in an annotated method, the appropriate advice is invoked.

Aspect-oriented programming can be beneficial when a new function should be added to multiple classes of an existing application with minimal effort and changes in the available code. Moreover, is it suitable when the classes are of a different kind, so as it is challenging to implement common super classes and inherit functions from them. The aspect-oriented approach could be potentially used to support creation of explorable XR environments by domain experts.

2.6 Problem statement

The available approaches to development of XR have the following limitations, which make them unsuitable for building explorable environments.

2.6.1 Explorable temporal 3D representation

The available semantic approaches do not enable explorable representation of temporal users' and 3D objects' properties with reasoning on and queries about different moments and periods in time. Reasoning allows content users to infer implicit (tacit) knowledge from explicitly specified knowledge. In turn, queries enable flexible access to users' and objects' properties using precise and complex conditions, while skipping properties irrelevant to a particular use case. Content exploration with reasoning and queries can be especially useful to develop shared (e.g., web-based) explorable XR environments and repositories of interactive 3D content.

Metadata frameworks in the available 3D formats and tools offer no classes, individuals, properties as well as relationships between them. Unstructured or semi-structured metadata is unsuitable for unambiguous reasoning as well as complex and precise queries including multiple conditions. It limits their expressiveness and accuracy, thereby narrowing possible practical applications.

2.6.2 Logging users' and objects' behavior

The available solutions do not enable logging of users' and objects' behavior, including actions and interactions that occur while using an environment. Logging users' and objects' behavior in a visual knowledge-based form could permit semantic exploration with reasoning and queries accompanied by visualization to allow users to watch and gain knowledge about interesting actions and interactions.

Information about users' and objects' behavior can be valuable in multiple application domains to acquire knowledge about users' experience, preferences and interests as well as to characterize objects in an environment. Methods and tools of logging behavior in XR should be flexible and efficient in covering arbitrary elements of the environment, which may be modules, classes, functions and variables, without introducing redundancy to the original code. Although different programming libraries are available for aspect-oriented programming in various languages, their potential has not been used to build explorable XR environments.

2.6.3 Domain knowledge-based representation

The available approaches do not enable flexible—applicable to different domains—connection of temporal users' and 3D objects' properties to domain knowledge. On the one hand, it limits possibilities of creating and using XR by domain experts, who rarely have technical skills. On the other hand, it reduces the possibilities of processing XR environments by systems that operate on domain knowledge but are incapable of extracting it from the environment.

Although object-oriented programming languages enable representation of 3D content with domain-specific concepts, such as classes and properties, it remains problematic to acquire knowledge from software modules. The syntax of object-oriented data structures, which indicates domain knowledge (e.g., hierarchies of classes, domains and ranges of properties), interlaces with imperative instructions, which specify steps to be completed rather than target domain-specific goals to be achieved. It makes reasoning and queries difficult and demanding explicit implementation of additional software functions. In turn, it typically requires advanced technical skills, which hinders contribution from domain experts, who are not IT specialists yet have domain knowledge that would be crucial at this stage. Moreover, special effort is needed to decouple the new functions responsible for knowledge processing from the previous functions responsible for 3D content processing, e.g., by refactoring the former code and applying apt design patterns. It may be time-consuming and expensive, especially when an environment is extended by programmers who have not developed it.

Taking into account reasoning and queries, object-oriented programming languages are much less suitable in comparison with declarative programming languages, description logics and the semantic web approach, for which widely known algorithms for reasoning and query processing as well as query languages are available.

3 Visual ontology-based behavior representation

The main contribution of this paper is an approach to *visual aspect-oriented modeling of explorable XR environments*, which addresses the issues and challenges mentioned in Sect. 2.6. The approach extends the ontology-based representation of interactions proposed in Flotyński and Sobociński (2018) by providing: a visual ontology-based representation of interactions and autonomous actions, visual aspect-oriented pipeline of modeling explorable XR as well as an implementation, evaluation and discussion of the overall approach with examples. The approach consists of three main elements:

1. The *visual ontology-based behavior representation*, which specifies terminology for visual semantic behavior logs. The representation addresses the challenge described in Sect. 2.6.1, and it is explained in this section.
2. The *logger ontology*, which is a data model for aspect-oriented modeling of explorable XR. The ontology addresses the challenge described in Sect. 2.6.2, and it is explained in Sect. 4.
3. The *pipeline of visual aspect-oriented modeling*, which is a development process that employs the logger ontology to produce XR environments that are explorable due to generating visual semantic behavior logs compatible with the behavior representation. The pipeline addresses the challenge described in Sect. 2.6.3, and it is presented in Sect. 5.

The *visual ontology-based behavior representation* is the pair of: a *domain ontology* and the *fluent ontology*, which are TBoxes encoded using the semantic web standards (RDF, RDFS and OWL—cf. Sect. 2.2). 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 the results of actions and interactions. Due to the use of the semantic web, 3D content behavior can be represented using general or domain knowledge.

1. A *domain ontology* specifies classes and properties related to a particular application domain with rela-

tionships between them, e.g., hierarchies, domains and ranges of properties as well as restrictions. A domain ontology is determined by a particular XR application and should be comprehensible to users or domain experts in the related field. A domain ontology is common to all use cases of the environment. Different domain ontologies may be used for behavior representations in different explorable environments, e.g., shopping in virtual stores, visiting virtual museums, planning virtual cities.

2. The *fluent ontology* specifies classes and properties derived from the 4D-fluents approach (cf. Sect. 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. In addition, the ontology specifies a link between temporal classes and visual descriptors of behavior, such as movies and images. The fluent ontology is an immutable part of the proposed approach, and it is common to all explorable XR environments.

A *visual semantic behavior log* is an ABox consisting of *temporal statements*. A temporal statement expresses an action or interaction using the terminology of the behavior representation, including its domain and fluent ontologies (Fig. 4). A temporal statement, which represents an action or interaction denoted by a *property* that links a *subject* and an *object*, consists of the following RDF triples:

1. (subject/object time slice, is instance of, time slice) and (subject/object time slice, is time slice of, subject/object), which defines the time slices of the *subject* and *object* according to the 4D fluents approach.
2. (subject time slice, property, object time slice), which combines both time slices using the *property*, which represents the action or interaction.
3. (subject/object time slice, has time, time interval/point), which provides information about the time when the action or interaction occurs, and whether it is temporary or instant.
4. (time interval/point, has visual descriptor, descriptor), which refers to a visualization of the action or interaction, e.g., a movie or images.

For instance, a temporal statement can incorporate: a *user* (subject) with its time slice linked to a time slice of an *appliance* by the *repairs* property. The reparation is lasting within a *time interval*, which links a *visual descriptor* of the interaction in the form of a movie. Temporal statements

Fig. 4 Temporal statement of visual semantic behavior log. Individuals and their properties marked in red (Color figure online)

that express actions have a similar structure, yet without all triples that include an object.

Hence, a behavior log represents a temporal state of affairs in a particular use (session) of an explorable XR environment. Reasoning engines can process logs to infer tacit (implicit) users' and objects' properties based on their explicit properties. Moreover, logs can be queried by users, applications and services to acquire visual and semantic information about actions and interactions.

4 Logger ontology for aspect-oriented modeling of explorable XR

The *logger ontology* is a TBox that specifies classes and properties describing how users' and objects' behavior is logged in explorable XR environments. The conceptualization of the logging process conforms to the aspect-oriented approach. The classes and properties of the ontology indicate which parts of the environment code should be extended and how to log environment behavior. The ontology is encoded using RDF, RDFS and OWL. Figures 5, 6 and 7 present the hierarchies of the classes, object properties and datatype properties visualized in the Protégé editor (Stanford University 2020), which has been used to develop the ontology.

Fig. 5 OWL classes specified in logger ontology

Fig. 6 OWL object properties specified in logger ontology

Fig. 7 OWL datatype properties specified in logger ontology

Using the semantic web to conceptualize the logging process in a structured way has the following advantages over the other semi-structured solutions such as JSON, XML and

XML Schema. First, it enables expressive representation of the terminology used, including classes and properties, with the common semantics established by the underlying standards. Second, it enables checking the consistency of logging process descriptions against the ontology.

4.1 Imperative code representation

The ontology specifies the organization of imperative code elements using the following classes and properties. The *application class* and *class method* represent classes and methods used in an XR environment. Methods are linked to classes using the *is method of* object property. Classes and methods may have *variables* linked by the *is variable of* object property. In addition, methods may have *parameters*, whose ordered RDF lists are linked by the *has parameters* object property. Every code element has a name, while a method and a variable also have a type.

4.2 Loggers

The overarching elements that specify how an XR environment is transformed to its explorable counterpart are *loggers*. A *logger* represents an aspect of logging an action or interaction in a behavior log, in line with the aspect-oriented approach (cf. Sect. 2.5). Every logger includes an RDF predicate that represents a feature, action or interaction. Different types of loggers are intended for logging different sorts of information about the environment and are applicable to behavior implemented in different ways. Furthermore, different types of loggers enforce the injection of logging code in different parts of the environment code.

4.2.1 Scene loggers

Scene loggers enable logging of immutable properties of the environment, which are set by a successful execution of a class method and do not change over time, e.g., landscape elements and buildings in a virtual tour. The *predicate* of a scene logger represents a fixed feature of the environment. Every logger includes a *logger point*, which represents an advice executed in a particular join point (according to the aspect-oriented approach). The use of different logger points by different types of loggers is depicted in Fig. 8. A logger point is linked to a logger using the *has logger point* object property. A logger point consists of a *subject* and a *literal value* or an *object*, which are linked using the *has logger subject*, *literal value* and *has logger object* properties, respectively. Subjects and objects are variables.

In addition, a logger point links a class method using the *has logger method* property. An execution of a class method captured by a logger injects a semantic statement on the subject, object and predicate to the behavior log. In the case

Fig. 8 Use of different logger points by loggers and impact on environment code

of a scene logger, the statement is time-independent and includes only triples that express the relationship between the subject and object (or literal) and their types (cf. Fig. 4). A scene point is an *on-exit point*, which means that logging is performed upon finalizing the class method execution. For example, a scene logger can impose that an invocation of a constructor of the building class logs the building as a permanent scene element with its position and other properties set by the constructor.

4.2.2 Behavior loggers

Behavior loggers enable logging of mutable properties of the environment, which change over time, e.g., the position, orientation and velocity of moving objects, and the structure of animated 3D scenes. The *predicate* of a behavior logger represents an action or interaction. Different types of behavior loggers are distinguished depending on the time of logging and the implementation of the behavior to be logged.

Time point loggers are used for logging instant actions and interactions, which occur at a point in time once a class method is successfully executed. Consequently, a time point logger has one logger point that is an on-exit point structured analogously to a scene point. Thereby, a time point logger imposes the injected logging code to be executed upon finalizing the method execution. Time point loggers determine logging that adds temporal statements with time points into the behavior logs (cf. Sect. 3). The value of the time point is determined by the *timestamp* datatype property set to a particular datetime or to the `now` keyword, which gets the current datetime during the method execution. For example, a time point logger states that an execution of the detach-transformer method logs the electrician who does it and the moment in time of the interaction.

Time interval loggers are used for logging temporal actions and interactions, which are occurring for a period of time. Therefore, they determine logging that adds temporal statements with time intervals into behavior logs.

Single method loggers are used for logging behavior that is implemented by a single method. Such a behavior starts at invoking a class method and finishes at the successful execution of this method. A *single method logger* has two logger

points: a *start point* that is an *on-enter point* and an *end point* that is an *on-exit point*. Hence, the logging of the behavior starts before executing the method and finishes upon finalizing the method. The start point has the timestamp set to *start*, which denotes the current datetime opening a new time interval. The end point has the timestamp set to *end*, which denotes the current datetime closing the time interval. Alternatively to the keywords, timestamps may be set to arbitrary subsequent moments in time if the behavior should be timed differently than it really happens, e.g., simulating historical or future events. A single invocation logger can be created, in particular, for a method that animates 3D objects for an interval of time, from the method invocation to its finalization.

If the behavior that should be logged starts at a successful execution of a class method and is lasting until the next successful execution of the same method, a *double invocation logger* is used. Hence, its start and end points are on-exit points. The logger point has timestamp set to *now* or to a particular datetime, which closes the previous time interval for the subject and opens a new interval, in which the predicate value for the subject may be different. For instance, in a VR house configurator, a room is painted a particular color from the moment of a color selection until the selection of another color—which is done by a single method in the environment.

If the behavior that should be logged starts at a successful execution of one class method and is lasting until the successful execution of another class method, a *double method logger* is used. It has two logger points, each indicating the final of a different method: a start point that is an on-exit point for the first method and an end point that is an on-exit point for the second method. For example, a customer in a VR store starts shopping upon executing the *start-shopping* method and finishes shopping upon executing the *finish-shopping* method.

5 Pipeline of modeling explorable XR environments

The *pipeline of modeling explorable XR environments* enables development of new explorable environments as well as transformation of existing environments to their explorable counterparts. The pipeline combines the capabilities of imperative object- and aspect-oriented programming with the semantic web, and it can be implemented in widely used software development environments, including game engines. Such a combination of technologies enables to address the requirements listed in Sect. 2.6 with regards to the following factors. First, the necessary changes of an existing environment should be minimal in terms of the additional code that must be prepared and

the previous code that must be modified. Second, the new code responsible for knowledge management should be decoupled from the original environment code. These are to minimize the programmers' effort, time spent and risk of undesirable side effects for the current presentation and behavior of the environment.

In the pipeline, modeling in the aspect-oriented manner is permitted by the *logger ontology*. The result of the pipeline—an explorable XR environment—logs users' and objects' actions and interactions in the form of *visual semantic behavior logs*. Behavior logs combine domain knowledge and visual descriptors of what happened while the environment was being used. Further, behavior logs can be subject to exploration covering reasoning and queries about states of the environment, with its users and objects, in different points and periods in time. The result of queries is knowledge about the requested behavior and its visualization.

A behavior log represents a particular *session*—period of using—an explorable XR environment, while users and 3D objects are acting and interacting. Only reported actions and interactions can be further explored and visualized in terms of involved participants as well as added, modified and removed 3D objects. The selection of behavior to be logged is a job of users of the environment, and it should cover use cases intended for knowledge exploration.

The pipeline consists of four steps: *developing a prototype XR environment and mapping it to domain classes*, *creating loggers for the environment*, *compiling the loggers* and *compiling the environment*. The steps are preceded by *designing a domain ontology*. All the steps, including the preceding one, are described in the following subsections and depicted in Fig. 9. In every step, created are components marked in red as well as completed are activities denoted by red arrows that are enclosed within the diagram area of the step as well as between the step and its previous step, e.g., specifying *instances*, *start* and *end methods* in Step 2.

The steps of the pipeline are accomplished using the *Visual Logger Editor (VLE)*. VLE is a plugin to MS Visual Studio, which consists of two components: the *logger manager* and *logger compiler*. The logger manager allows users to create mappings and loggers in Steps 1–2. The logger compiler compiles loggers, which is followed by the injection of code responsible for logging the environment behavior while it is running. The compiler is used in Step 3. Both components employ the Open-Link Virtuoso triplestore (Open-Link 2020) to store the behavior representation ontologies, the logger ontology and behavior logs. They communicate with the triplestore and process ontologies and logs using the dotNetRDF library (dotNetRDF 2020). The architecture and data flow of VLE are depicted in Fig. 10. The VLE GUI has been implemented using the XAML GUI description language, and it is presented in Fig. 11.

Fig. 9 Pipeline of visual aspect-oriented modeling of explorable XR environments (Color figure online)

Fig. 10 Architecture and data flow of Visual Logger Editor

Fig. 12 Example of domain ontology for servicing home appliances. Visualization of OWL graph in Protégé OntoGraf

5.1 Designing domain ontology

Designing a domain ontology is the step the precedes the steps of the modeling pipeline. Domain classes and properties are used in the pipeline to represent users' and objects' 3D behavior in a way comprehensible to domain experts, who typically are the authors of the ontology and the main users of the XR environment. Domain ontologies may be created using ontology engineering tools, such as Protégé (Stanford University 2020). A fragment of a domain ontology for servicing home appliances is presented in Fig. 12. It includes entities representing appliances and interactions of users with the appliances while servicing. The nodes represent OWL domain classes with the RDFS *sub-class of* relationships: *user*, *appliance* (with the subclasses *electric cooker* and *induction hob*) and *element* (with the subclasses *fan*, *display*, *heating plate* and *coil*). The arcs present OWL domain properties: the *includes* relationship

Fig. 11 Visual Logger Editor in MS Visual Studio

between *appliances* and their *elements* as well as interactions between *users* and *elements* (*assemblies*, *disassemblies* and *tests*).

Designing domain ontologies should respect the anticipated use cases of the environment and knowledge exploration. This process has been thoroughly described in the literature (e.g., Gayathri and Uma 2018; Baset and Stoffel 2018; Pouriyeh et al. 2018; Mkhinini et al. 2020) and is out of the scope of this paper. Once a domain ontology is ready to use, it can be loaded into VLE, including classes as well as object and datatype properties (the bottom listviews in Fig. 11).

5.2 Step 1: developing prototype XR environment and mapping to domain classes

In this step, a *prototype XR environment* (an XR application) is developed using an imperative programming language (e.g., Java or C#), possibly with a game engine (e.g., Unity or Unreal). Its code consists of *application classes*, which include *class variables* and *class methods*. Every class method may include *parameters* of invocation and *local variables*. Such entities are represented using code elements of the logger ontology (cf. Sect. 4.1). A prototype environment is created by developers, including programmers and graphic designers. The proposed approach is devised for both environments intentionally developed for exploration as well as environments that were originally developed for other goals.

VLE can be started for any solution in Visual Studio. It reads from the solution all application classes with their class variables and methods as well as the parameters and local variables of the methods and presents them hierarchically in a tree view (left panel in Fig. 11).

5.2.1 Mapping in Visual Logger Editor

Further, application classes of the prototype environment are mapped to domain classes of the domain ontology. It enables knowledge-based representation of the terminology used in the environment (classes of users and 3D objects) by the terminology of the domain ontology. It is finally reflected in behavior logs in the assertions on individual objects that belong to the mapped domain classes. An example of creating a mapping with the VLE logger manager is presented in Fig. 13. Every *mapping* is a triple (application class URI, mapping, domain class URI). Hence, the mapping RDF property has domain equal to the application class and range equal to the owl:Thing class.

Mappings are created by domain experts who have basic knowledge about the foundations of the object-oriented data model (cf. Sect. 8). Mappings for a particular environment and a particular domain ontology are stored in an ABox

Fig. 13 Mapping application classes of XR environment to domain classes of domain ontology

referred to as a *mapping knowledge base*. An example of a mapping knowledge base is presented in Listing 1. The bridge between the object-oriented data model and semantic data model provided by mappings permits further exploration of the environment scene and behavior, including reasoning and queries about users and 3D objects represented by domain classes.

Listing 1 Example of mapping between application class and domain class in RDF Turtle format.

```
1 @prefix mkb: <http://semantic3d.org/aspects/
   ImmersiveServiceGuide/mapping#> .
2 @prefix env: <http://semantic3d.org/aspects/
   ImmersiveServiceGuide#> .
3 @prefix do: <http://semantic3d.org/aspects/
   household-ontology#> .
4 @prefix rdf: <http://www.w3.org/1999/02/22-
   rdf-syntax-ns#> .
5
6 mkb:mapping rdf:type rdf:Property .
7 env:Appliance mkb:mapping do:Appliance .
```

5.2.2 Guidelines for developers

Although the approach can be used for various XR environments, developed with imperative languages in different ways, some development guidelines are advised. Processing loggers injects new code instructions at the beginning and at the end of the class methods captured by the loggers. It requires that the variables used by the loggers are accessible in the form in which they should be logged. The following guidelines are for developers who use the proposed 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 loggers will refer should be declared in the main scope of the method but not in narrower scopes, e.g., within conditional and loop instructions.

- Local method variables whose single execution—from start to finish—reflects a process (e.g., animation), which can be logged, should be assigned once declared. Such methods can potentially be used by single invocation loggers, which start logging once starting a method.
- A method should finalize with up-to-date values of the variables to be logged, which should be equal to the values used in the method. The preparation of the variable values should be done at the beginning of the method.

Although the guidelines introduce certain restrictions on how the approach is used, developers may apply them 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.

- If a logger refers to a local method variable that is not defined in the main scope of the method, the variable declaration is moved to the main scope.
- 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 methods while leaving the variables unchanged at the end of the primary method.
- If a method, after completing its main job, is invoking other methods:
  - Copies of the variables to be logged should be passed to the methods instead of references to the variables. It prevents modification of the variables by the other method invoked, which would lead to logging invalid values, or
  - 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.

- If it is necessary to log actions and interactions from methods implemented in external libraries that are available only as binaries, the methods should be wrapped by additional methods with loggers. For instance, if an environment invokes a web service to get a new 3D object for a scene, which should be logged, the invocation is moved to an extra method, which will be captured by a logger, thereby logging the execution of the service.

5.3 Step 2: creating loggers for prototype XR environment

In this step, a *logger knowledge base* is created. A *logger knowledge base* is an ABox in which individual loggers are specified using terminology (logger classes and

properties) provided by the *logger ontology* (cf. Sect. 4). Creating an example of a *time point logger* with the VLE logger manager is depicted in Fig. 11 (the *Create behavior logger* dialog). The logger captures an instant interaction, which is the assemblage of an appliance element by a user. The interaction should be logged upon executing the *assemble* method. Since the method belongs to the user class, the *this* keyword indicates the subject as the user for whom the method is invoked. The logger predicate is *assembles*, and the object is the *appliance element* method parameter. Only one logger point is assigned to a time point logger as it captures an instant interaction, which occurs at a moment in time. Therefore, the other textboxes in the dialog remain empty.

The created logger is visualized in Fig. 14. Classes have been skipped to make the scheme more clear. The logger captures an instant interaction at a moment when it happens, which is determined by the *now* timestamp. The point indicates the *assemble* logger method as well as the subject (*this*) and object (*appliance element*) involved in the interaction. In a temporal statement logged by the method execution, the subject and object will be linked by the *assembles* predicate, which is assigned to the logger using the OWL datatype property. In a similar, visual way, other loggers are created, e.g., a logger that also captures the *assemble* method to log that the appliance element starts to be included in an appliance since assembled. Such a pair of loggers listed in VLE are depicted in Fig. 15. Both loggers will capture the exit of the *assemble* method.

Fig. 14 Time point logger created with Visual Logger Editor. Visualization of OWL graph in Protégé OntoGraf

Behavior Loggers

```
OnExit AspectModeling.Class.Assemble (this , http://semantic3d.org/aspects/household-ontology#assembles , applianceElement)
OnExit AspectModeling.Class.Assemble (appliance , http://semantic3d.org/aspects/household-ontology#includes , applianceElement)
```

Fig. 15 Behavior loggers presented in Visual Logger Editor

5.4 Step 3: compiling loggers

In this step, the VLE logger compiler processes the entire VS solution and completes the following actions.

1. It attaches the *log library* to the solution. The library implements an algorithm that generates behavior logs while the environment is being used. The algorithm has been explained in Flotyński and Sobociński (2018). It generates temporal statements based on loggers and loads the statements to the triplestore.
2. It injects instructions that are capturing the screen while the XR environment is running and save the data as *visual descriptors*, which individual images or movies. Images, which are screenshots, are captured using the method `ScreenCapture.CaptureScreenshot` in Unity (Unity 3D 2020). Screen capturing is implemented in the same thread as rendering, together with logging the behavior. Thus the captured images strictly reflect the actions and interactions occurring in the 3D scene. Further, movies are generated from the images using FFmpeg (2020). Visual descriptors are linked to time points and time intervals of the temporal statements as explained in Sect. 3 and Fig. 4.
3. It extends the body of every class method captured by a logger with invocations of appropriate log library methods. An invocation is injected at the beginning or end of the captured class method, depending on the logger type. The scheme of a class method with injected code is presented in Fig. 8. The injected instructions are responsible for extracting information about the classes of the variables used by the loggers, creating timestamps, setting identifiers of the semantic individuals to be inserted into the log, and invoking methods from the log library that build and insert the temporal statements to the triplestore. The input parameters of the log library methods are the variables and literal values given in the loggers, the extracted classes of the variables and created timestamps.
4. It injects additional common classes and methods responsible for setting values of temporary variables used in the code inserted in the previous actions.

5.5 Step 4: compiling XR environment

In this step, a native compiler specific to a particular hardware and software platform (e.g., the VS C# compiler) is used to generate the explorable XR environment in its executable form, which results from the pipeline. The explorable XR environment comprises all the classes with the methods and variables from the prototype environment and the logging code injected in Step 3. Hence, its 3D objects appear and behave like their prototypes, yet the class methods captured by loggers created in Step 2 and compiled in Step 3 also generate behavior logs while the environment is used. Behavior logs conform to the behavior representation explained in Sect. 3.

6 Behavior logs in explorable immersive service guide

In this section, the process of transforming an XR environment to the explorable form and the process of generating and using behavior logs are presented.

6.1 Immersive service guide for home appliances

An immersive service guide for home appliances has been developed in the virtual reality laboratory at the Poznań University of Economics and Business in Poland (Flotyński et al. 2019b). The guide has been developed to train technicians in servicing and using induction hobs of Amica S.A., which is one of the main producers of household equipment in Poland.

The guide is a Unity-based application. Presentation of 3D models of products to users is enabled by an HTC Vive headset. The headset enables immersive visualization of 3D content at a level of detail which is sufficient for step-by-step training in repairing defects and testing home appliances. Immersive visualization enhances the user's experience in comparison with more traditional 2D presentation. Interaction with 3D models of induction hobs is permitted by a Leap Motion device, which is capable of tracking user's hand gestures. Different hand gestures

Fig. 16 Testing power connection of induction hob

are used to perform different actions with hobs, e.g., disassembling, assembling and testing particular elements. Animated users' hands are presented within the XR scene using realistic 3D models. It is an intuitive, user-friendly form of interaction. The used devices form an affordable platform, which is attainable to small distributors of household appliances as well as average users.

The Unity game engine has been selected because it supports multiple XR devices and has extensive documentation with a lot of tutorials. It has enabled the integration of the aforementioned equipment into the immersive environment with reasonable effort.

The guide implements several immersive scenarios of repairing the most common defects and testing induction hobs. The guide allows users to look at different elements of a hob represented with high accuracy, zoom them in and out, and watch animations and movies that present activities to be performed. The guide covers the following immersive interactive 3D scenarios.

1. *Testing the power connection* (Fig. 16). This scenario is devoted to one of the most common failures. In an animation, the appropriate terminals are indicated, and their correct connection is shown. Finally, the voltage on the connection is measured using a virtual multimeter.
2. *Testing transistors and display* (Figs. 17, 18). This scenario demonstrates how to verify the transistors and display, which are keys to the correct work of the appliance. Once verified, the elements may be disassembled if they are broken, or the work of the hob may be tested in other scenarios.
3. *Disassembling and assembling coils, heating plates and the fan* (Figs. 19, 20, 21). These scenarios are executed when coils, heating plates or the fan are broken and need to be exchanged. The scenarios allow service techni-

Fig. 17 Testing transistors of induction hob

Fig. 18 Testing display of induction hob

Fig. 19 Disassembling and assembling coils of induction hob

Fig. 20 Disassembling heating plates of induction hob

Fig. 21 Disassembling fan of induction hob

Fig. 22 Testing work of induction hob

cians to repair the hob, which can be verified in another scenario.

4. *Testing the overall work of the hob* (Fig. 22). This scenario allows users to test powering on the hob and cooking with it. Possible problems that may occur during the operation of the hob may be presented together with solutions, or correct work of the appliance is reported.

Fig. 23 Graphical representation of visual semantic behavior log generated while using immersive service guide

6.2 Transformation to explorable service guide

The aforementioned interactive scenarios of the guide are implemented using distinct class methods, in particular *disassemble*, *assemble* and *test* of the *user* class presented in the left panel of VLE in Fig. 11. The guide was loaded into VS and processed with VLE, including all the steps of the pipeline, to generate the explorable XR service guide. First, the classes *user*, *appliance* and *appliance element* were mapped to the corresponding domain classes using the VLE logger manager in Step 1 of the pipeline. Second, the class methods were captured by loggers using the logger manager in Steps 2 of the method. Third, the loggers were compiled by the VLE logger compiler in Step 3. Finally, VS was used to compile the entire XR environment and generate its executable explorable form in Step 4.

6.3 Example of behavior log

A behavior log was generated while a serviceman was training with the explorable immersive service guide. A fragment of the log is presented in Fig. 23. The log presents past behavior of a serviceman, which encompasses his

interactions with elements of an induction hob. The interactions are implemented within different scenarios by class methods (Sect. 6.1). The log is an ABox, which uses classes and properties specified in the fluent ontology and the *induction hob service ontology*. The terminology of the latter is comprehensible to service technicians. Both ontologies form the behavior representation for the service guide.

Every node in the log graph denotes an OWL individual, which falls into one of the three main groups: time points, time intervals and visual descriptors; time slices of domain objects; and domain objects. The corresponding classes specified in the behavior representation have been skipped in the log graph.

The log consists of four temporal statements, which express consecutive interactions and the states following them. The statements are built according to the scheme depicted in Fig. 4. First, a user (serviceman) disassembles an old coil of the hob. This is an instant interaction, which occurs at a point in time linking an image visual descriptor captured at the point. The disassemble property, which represents the interaction, links the time slices of the user and the old coil. The disassembly closes the time interval when the old coil was a part of the hob, which was expressed by the *includes* property in temporal statement 2. The statement incorporates a movie visual descriptor which captures all frames from the environment that were within the interval. Next, the user assembles a new coil, which also happens at a point in time and has an image as the visual descriptor. This interaction, in turn, starts the new interval in time when the hob includes the new coil (temporal statement 4). This temporal relationship is visualized by a movie.

6.4 Querying behavior log

Exploration of actions and interactions in the immersive service guide is possible with queries to the generated behavior logs. Queries may be encoded in SPARQL (W3C 2013). Such exploration can provide information about how to accomplish different service activities, which can be used to teach beginner technicians.

An example of a SPARQL query to the generated behavior log prepared and executed using the Open-Link Virtuoso web interface is presented in Fig. 24. The query searches for information about assembling a coil. The result set contains values for all the variables used in the query, including: the user who assembled the coil, the assembled coil, the point in time when the interaction occurred and the visual descriptor of the interaction. The point in time is absolute. However, the relative time point can be calculated by subtracting the absolute time of starting the environment from the requested time point. The visual descriptor

Fig. 24 Example of query to semantic behavior log in Open-Link Virtuoso

is a single image (presented at the bottom of Fig. 19) as the interaction was instant.

7 Evaluation

The approach has been evaluated in terms of: the size of the data structures used and code generated while modeling and logging as well as the performance of processing the data structures.

7.1 Processing performance

The performance of VLE was evaluated in terms of compiling loggers, which is an essential step of modeling explorable XR environments. In addition, the performance of the log library responsible for logging behavior in such environments was evaluated. The tests have been completed using the following work stations connected by the 10 Gigabit Ethernet. *Work station 1* was 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. *Virtual work station 2* was equipped with 2 CPU AMD EPYC 7551 32-Core 2.0 GHz with 32 cores and 64 threads; 8 GB RAM DDR; and HD 160 GB.

7.1.1 Compiling loggers

The time of compiling loggers by the VLE logger compiler at works station 1 was evaluated. Logger knowledge bases including from 50 to 750 loggers of a particular type were generated. Five loggers were assigned to a single class method. Hence, the number of methods was varying in the range 10–150 for all types of loggers except double method loggers, which captured twice more methods. For every type and every number of loggers, 20 different logger knowledge bases were generated and used to test the logger compilation.

Fig. 25 Performance of logger compilation

The compilation time is presented in the graph and table in Fig. 25. Every point in the curves represents the average time for 20 compilations. The compilation time varies from 10 to 20 ms (for 50 loggers applied to 10 or 20 methods) to about 600–2000 ms (for 750 loggers applied to 150 or 300 methods). The formulas calculated using interpolation as well as the coefficient of determination (R^2) equal almost to 1 show that the compilation time is proportional to the square of the number of loggers. The most time-consuming is the compilation of double method loggers as it injects code into different methods. Also, the compilation of single invocation loggers is relatively time-consuming as it injects code at the beginning and end of methods. The compilation of time point loggers, double invocation loggers and scene loggers is the most efficient as it affects only one method in its one point. The relation between the curves in the graph is confirmed by the calculated average time of compiling a single logger.

7.1.2 Logging behavior

The performance of the log library was evaluated. The experimental explorable environment was streaming the generated temporal statements. The behavior was an animation of a moving object that lasted for 10 min. The tests were performed for 2 configurations that differ in terms of the location of the triplestore: at the localhost at work station 1 (together with the environment), and on the remote host (work station 2) in the same Campus Area Network (CAN). In every update of the animated object, temporal statements about its position were generated and transmitted to the triplestore. For every configuration of the test environment, streaming was repeated 20 times. The average results are summarized in the table in Fig. 26. Whereas inserting

	Criteria	Localhost	Campus area network	CAN / Localhost
Time [ms]	Average	26,35	67,19	2,55
	Standard deviation	1,88	9,54	5,08
	Coefficient of variation	0,07	0,14	1,99
	Minimum	110,69	273,91	2,47
	Maximum	183,98	864,48	4,70
Number of	Temporal statements	21 000	8 005	0,38
	Triples	78 856	31 858	0,40
	Objects	13 146	5 313	0,40
Size	Temporal statement [B]	234,71	247,43	1,05
	Log [KB]	4 813,39	1 934,27	0,40

Fig. 26 Performance of logging behavior on localhost and over CAN network with complexity and size of generated logs

statements to the triplestore at the localhost required 26 ms, the performance for the CAN was 2.55 times lower. It denotes the average network delay 41 ms. The standard deviation divided by the average (coefficient of variation) is twice larger for the CAN, which means the less stable transmission. It corresponds to the increasing ratio between the minimum and maximum values for both configurations. As rendering and logging were executed within the same thread, the upper limit of FPS was 38 for the localhost and 15 for the CAN.

7.2 Complexity and size of data structures

The evaluation also encompasses the complexity and size of data structures used in the consecutive steps of the pipeline and generated while using explorable environments:

- (1) mapping knowledge bases created in Step 1,
- (2) the logger ontology and logger knowledge bases created in Step 2,
- (3) the environment code before and after the logger compilation in Step 3,
- (4) behavior logs generated while using explorable environments.

7.2.1 Mapping knowledge bases

The VLE logger manager generates mappings in the RDF Turtle format (World Wide Web Consortium 2014). In the explorable service guide, the size of the mapping knowledge base is equal to 696 B. The part that is common to every mapping knowledge base has size 318 B. The rest—specific part—includes mappings of 9 application classes to domain classes (presented in Fig. 12). Hence, the average size of a single mapping is 42 B.

7.2.2 Logger ontology and knowledge bases

The numbers of different entities in the logger ontology, which is used by the VLE logger manager to create loggers, have been summarized in the table in Fig. 27. The ontology

Concepts	Number
classes	27
datatype properties	7
object properties	12
qualified cardinality restrictions	17
has value restrictions	2
triples	238

Fig. 27 Complexity of logger ontology

Loggers				Injected code		
Logger type		Logger size [bytes]	Logger triples	Class code [bytes]	Method code [bytes]	Overall code size [bytes]
Behavior logger	Time point logger	1161	29	1859	425	2284
	Single method logger	2120	52	1859	857	2716
	Double invocation logger	2119	52	1859	425	2284
Scene logger	Double method logger	2115	52	3718	850	4568
		1126	28	1859	296	2155

Fig. 28 Size of loggers and injected code

includes classes and properties presented in Figs. 5, 6 and 7. Qualified cardinality restrictions (typically *OWL exact cardinality*) are used to determine the structure of loggers, e.g., a time point logger has exactly one logger point. *OWL has value* restrictions are used to determine the values of timestamps for start and end logger points. The overall number of RDF triples in the ontology is 238, while its overall size is equal to 18,9 KB.

Also, the size of logger knowledge bases that include one logger of a particular type (Fig. 28) was evaluated. The largest is the size of a single invocation logger which, however, is similar to the size of a double invocation logger and a double method logger. The reason is the same structure of such loggers, which includes two logger points. This is reflected by the same number of triples. Time point loggers and scene loggers are significantly smaller because of including only one logger point. The difference between them is minor as time point loggers include one additional triple that expresses a timestamp of a logger point.

7.2.3 Code injected into XR environments

Compiling loggers in Step 3 of the pipeline is followed by the injection of logging code into the environment. The analysis encompasses the relation between the size of a single logger and the size of the logging code injected into the classes and methods by the VLE logger compiler (Fig. 28). Taking into account the overall injected code, which includes the code extending a class and the code extending a class method, the largest is the code injected while compiling double method loggers. Such code spreads across two methods of some (possibly different) classes, which are responsible

Fig. 29 Size of time point loggers as well as application classes before and after logger compilation

Fig. 30 Size of single invocation loggers as well as application classes before and after logger compilation

for starting and finishing logging. Although a single invocation logger affects only one class, which reduces the class-specific code, the method-specific code is almost the same as in the case of a double method logger, because every method is extended in two points. The code imposed by the other loggers is much smaller and almost equal for time point loggers, double invocation loggers and scene loggers. It is because a single method of a class is extended while compiling such loggers. The slightly smaller code in case of scene loggers is caused by the lack of instructions that compute timestamps. Although the overall size of the injected code is larger than the size of a single logger, the majority (about 80%) is constituted by the code injected at the class level, which is commonly used by the code injected into different class methods. The method-specific code constitutes 13–32% of the overall code imposed by a logger.

The overall size of logger knowledge bases was compared with the size of the captured classes of an environment before and after the compilation. Only the class template and the code injected by the compilation of loggers have been taken into account, regardless of the original class code. The used knowledge bases include from 50 to 750 loggers of different types. The results are presented in Figs. 29, 30, 31, 32

Fig. 31 Size of double invocation loggers as well as application classes before and after logger compilation

Fig. 32 Size of double method loggers as well as application classes before and after logger compilation

Fig. 33 Size of scene loggers as well as application classes before and after logger compilation

and 33. The classes before the compilation were the same in all the cases. The largest is the size of logger knowledge bases that consist of time interval loggers: single invocation, double invocation and double method loggers, which have two logger points. It is about twice larger than the size of time point and scene loggers, which have one logger point. Logger knowledge bases are about 8–16 times larger than

the generated classes, which in turn are 4–8 times larger than the classes before the compilation.

7.2.4 Behavior logs

The size of the behavior logs generated by streaming temporal statements (cd. Sect. 7.1.2) is summarized in the table in Fig. 26. The log generated on localhost has size 4813 KB and is more than twice larger than the log for the CAN, which strictly corresponds to the time of inserting a single temporal statement to the triplestore for both configurations. This proportion spreads on the numbers of temporal statements in the logs and, consequently, the numbers of RDF triples and objects in the logs. The logged temporal statements were of similar size in both cases.

8 Discussion

8.1 Evaluation results

The obtained results show that the approach can efficiently transform XR environments to their explorable counterparts. The time of compiling environments on a moderately efficient computer varies from several milliseconds, for environments with dozens of captured methods, to about 2000 milliseconds, for environments with hundreds of captured methods. Furthermore, the polynomial formulas computed with high accuracy show that the compilation time does not rapidly grow with the increase in the size of the logger knowledge base. Thus, the performance of compilation can be considered acceptable to individual users as well as developers of web services—to enable the on-demand transformation of XR environments.

The performance of logging behavior strongly depends on the relative location of the explorable environment and its triplestore. It is highest in the case of the environment and triplestore installed on the same host, which enables logging animations, while maintaining acceptable FPS. The logging time is much higher when temporal statements are transmitted over a network, which significantly decreases FPS. In such a case, the approach is appropriate for logging occasional events rather than animations. This problem can be solved in different ways. First, logging behavior can be performed by a different thread than the one responsible for rendering the 3D scene. In such a case, temporal statements can wait in a FIFO queue for saving to the triplestore, without stopping the primary graphical thread. Second, the transmission can gather multiple temporal statements into larger packages to reduce the average delay. Third, other triplestores should be tested in terms of efficient network communication.

The size of mapping knowledge bases is small compared to the other datasets used in the approach, and it should not matter to any practical applications. Likewise, the size of the logger ontology is moderate. Hence, it is suitable for practical usage.

The behavior logs generated in the analyzed use cases are of acceptable size, taking into account possible practical applications of the approach, such as training, marketing and merchandising, even in cases of streaming annotations to a triplestore. Thus, the approach addresses the challenges described in Sects. 2.6.1 and 2.6.2. However, the structure the logger knowledge bases generated by Virtuoso seems to be verbose when compared with classes generated after compiling loggers. Encoding loggers in a more compact form would be beneficial for creating repositories of logger knowledge bases. A solution can be to organize and collect Turtle triples for common subjects and common predicates, which can significantly reduce the final document size. Also, RDF/JSON (World Wide Web Consortium 2013) and JSON for Linked Data (JSON-LD) (World Wide Web Consortium 2020) can be tested as possible formats for logger knowledge bases.

8.2 Aspect-oriented programming libraries

The current implementation of the VLE logger compiler is based on classes from the `System.Text.RegularExpressions` namespace in .NET. Aspect-oriented libraries [e.g., PostSharp (SharpCrafters 2020)] were considered as an alternative. The use of such a library could facilitate the structure and code of the logger compiler. Moreover, it would enable a relatively simple distinction of throwing exceptions by methods captured by loggers, which will be addressed in the approach. However, according to the author's best knowledge, currently no aspect-oriented libraries integrate with Unity, which is used in the projects.

8.3 Requirements for technical knowledge and skills

The author intends to facilitate development of explorable XR environments by average users and domain experts, who typically have limited programming and 3D modeling skills (the challenge discussed in Sect. 2.6.3). The main requirement for using the approach is the knowledge of the basic assumptions of the object-oriented data and processing model: classes, methods and variables as well as the RDF data model based on triples. Such knowledge is sufficient to use VLE. In addition, the concepts are used in a user-friendly visual way. The other requirement is the knowledge of the domain ontology used and the environment itself. The selection of class methods, variables and parameters for predicates, subjects and objects of loggers

can be supported by data flow and activity diagrams (e.g., in UML). Thereby, the approach can also be used by project managers and users familiar with design tools. Technical support from programmers is necessary if the prototype XR environment must be refactored before the transformation as explained in Sect. 5.2.2.

9 Conclusions and future works

In this paper, the approach to visual modeling of explorable XR environments is proposed. In such environments, actions and interactions of users and 3D objects are logged in a way that permits their further temporal exploration with semantic reasoning and queries combined with visual exploration. The approach enables creation of explorable XR environments based on available environments. The modeling process is accomplished visually with emphasis on high-level (domain-oriented) semantics of the intended environment. It can facilitate using the approach by people without programming skills, in particular domain experts, who should be critical participants of developing XR for their application domains, but typically lack advanced technical skills. Therefore, the proposed approach can improve the overall dissemination of XR.

The future works encompass several possible directions. First, the implementation of network communication needs to be improved to enable logging animations, which stream temporal statements to triplestores. Other existing libraries and triplestores can be evaluated, and critical components can be optimized. Second, the implementation can be extended in case of the availability of aspect-oriented libraries compatible with game engines. Third, the fluent ontology can be exchanged with the OWL-Time ontology (World Wide Web Consortium 2020), which provides comprehensible terminology related to time, including instants (time points) and intervals. It would align the time representation with the leading candidate for standardization. Next, the concept of explorable XR environments can be extended to semantically configurable XR environments. Such environments could permit query-based modification of the application state at any point in time and launching the environment from the altered state. The approach could also be evaluated by users creating explorable XR environments with the implemented development tools, e.g., in terms of the required time and effort. Finally, explorable XR environments can be applied in other domains. In particular, the temporal ontology-based representation can enable observation of the learning process and generation of personalized feedback to students.

Acknowledgements The author would like to thank Anna Englert, Mikołaj Maik and Adrian Nowak—students at the Poznań University

of Economics and Business, for the implementation of the immersive service guide used as an illustrative example in this paper. The author would also like to thank Amica S.A. for supporting the development of the immersive service guide and providing 3D content models and graphics of products.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- Artale A, Franconi E (2001) A survey of temporal extensions of description logics. *Ann Math Artif Intell* 30(1–4):171–210. <https://doi.org/10.1023/A:1016636131405>
- Autodesk (2020a) 3ds Max. <https://www.autodesk.pl/products/3ds-max/overview>
- Autodesk (2020b) Motion Builder. <https://www.autodesk.com/products/motionbuilder/overview>
- Baader F, Calvanese D, McGuinness DL, Nardi D, Patel-Schneider PF (2010) *The description logic handbook: theory, implementation and applications*, 2nd edn. Cambridge University Press, Cambridge
- Baader F, Horrocks I, Lutz C, Sattler U (2017) *An introduction to description logic*. Cambridge University Press, Cambridge. <https://doi.org/10.1017/9781139025355>
- Baset S, Stoffel K (2018) Object-oriented modeling with ontologies around: a survey of existing approaches. *Int J Softw Eng Knowl Eng* 28(11n12):1775–1794
- Batsakis S, Petrakis E, Tachmazidis I, Antoniou G (2009) Temporal representation and reasoning in OWL 2. *Semant Web* 8:1–20
- Ben Ellef 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. In: *Underwater 3D recording and modeling ISPRS*, Limassol, Cyprus
- Berners-Lee T, Hendler J, Lassila O (2001) The semantic web. *Sci Am* 284(5):34–43
- Blender Foundation (2020) Blender. <http://www.blender.org>
- Chmielewski J (2008) Interaction interfaces for unrestricted multimedia interaction descriptions. In: *Proceedings of the 6th international conference on advances in mobile computing and multimedia*, ACM, New York, NY, USA, MoMM '08, pp 397–400
- Chmielewski J (2012a) Describing interactivity of 3D content. In: Cellary W, Walczak K (eds) *Interactive 3D multimedia content*. Springer, Berlin, pp 195–221
- Chmielewski J (2012b) Finding interactive 3D objects by their interaction properties. *Multimedia Tools Appl* 69:1–26
- Chu Y, Li T (2008) Using pluggable procedures and ontology to realize semantic virtual environments 2.0. In: *Proceedings of The 7th ACM SIGGRAPH international conference on virtual-reality continuum and its applications in industry*, ACM, New York, NY, USA, VRCAI '08, pp 27:1–27:6
- Chu Y-L, Li T-Y (2012) Realizing semantic virtual environments with ontology and pluggable procedures. In: *Applications of virtual reality*, vol 171. <https://doi.org/10.5772/36761>
- De Troyer O, Kleinermann F, Mansouri H, Pellens B, Bille W, Fomenko V (2007a) Developing semantic VR-shops for e-Commerce. *Virtual Reality* 11(2–3):89–106
- 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 international conference on conceptual modeling—ER 2007*, Auckland, New Zealand, CRPIT, vol 83, pp 3–18
- 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
- dotNetRDF (2020) dotNetRDF—an Open Source .NET Library for RDF. <https://www.dotnetrdf.org/>
- Drap P, Papini O, Sourisseau JC, Gambin T (2017) Ontology-based photogrammetric survey in underwater archaeology. In: *European semantic web conference*. Springer, Berlin, pp 3–6
- FFmpeg (2020) A complete, cross-platform solution to record, convert and stream audio and video. <https://www.ffmpeg.org/>
- Flotyński J (2020) Knowledge-based explorable extended reality environments. Springer International Publishing, Cham. <https://doi.org/10.1007/978-3-030-59965-2>
- Flotyński J, Sobociński P (2018) Logging interactions in explorable immersive VR/AR applications. In: *2018 International conference on 3D immersion (IC3D)*, Brussels, 5–6 Dec 2018. IEEE, pp 1–8. <https://doi.org/10.1109/IC3D.2018.8657830>
- Flotyński J, Walczak K (2014) Semantic representation of multiplatform 3D content. *Comput Sci Inf Syst* 11(4):1555–1580
- Flotyński J, Walczak K (2015) Ontology-based creation of 3D content in a service-oriented environment. In: *Lecture notes in business information processing, 18th international conference on business information systems*. Springer, Berlin
- 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>
- Flotyński J, Brutzman D, Hamza-Lup FG, Malamos A, Polys N, Sikos LF, Walczak K (2019a) 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
- Flotyński J, Strugała D, Walczak K, Englert A, Maik M, Nowak A, Jenek T (2019b) An immersive service guide for home appliances. In: *2018 IEEE 8th International conference on consumer electronics—Berlin (ICCE-Berlin)*, IEEE Xplore, pp 370–375. <https://doi.org/10.1109/ICCE-Berlin47944.2019.8966215>. <https://ieeexplore.ieee.org/document/8966215>
- Games E (2020) Unreal engine. <https://www.unrealengine.com/>
- García-Rojas A, Vexo F, Thalmann D, Raouzaïou 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
- Gayathri R, Uma V (2018) Ontology based knowledge representation technique, domain modeling languages and planners for robotic path planning: a survey. *ICT Express* 4(2):69–74. <https://doi.org/10.1016/j.ict.2018.04.008>. (SI on Artificial Intelligence and Machine Learning)
- Gruber T (2009) Encyclopedia of database systems. <http://tomgruber.org/writing/ontology-definition-2007.htm>
- Gutierrez C, Hurtado C, Vaisman A (2005) Temporal RDF. In: Gómez-Pérez A, Euzenat J (eds) *The semantic web: research and applications*. Springer, Berlin, pp 93–107

- Gutiérrez M, García-Rojas A, Thalmann D, Vexo F, Moccozet L, Magneat-Thalmann N, Mortara M, Spagnuolo M (2007) An ontology of virtual humans: incorporating semantics into human shapes. *Vis Comput* 23(3):207–218
- 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/>
- 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>
- Kalogerakis E, Christodoulakis S, Moutoutzis 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
- Klein M, Fensel D (2001) Ontology versioning on the semantic web. In: Proceedings of the first international conference on semantic web working, CEUR-WS.org, Aachen, Germany, SWWS'01, pp 75–91. <http://dl.acm.org/citation.cfm?id=2956602.2956610>
- Kleiner F, De Troyer O, Mansouri H, Romero R, Pellens B, Bille W (2005) Designing semantic virtual reality applications. In: Proceedings of the 2nd INTUITION international workshop, Senlis, pp 5–10
- Lugrin JL (2009) Alternative reality and causality in virtual environments. PhD thesis, University of Teesside, Middlesbrough, United Kingdom
- 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, vol 2351, pp 282–292. International Society for Optics and Photonics
- Mkhinini MM, Labbani-Narsis O, Nicolle C (2020) Combining UML and ontology: an exploratory survey. *Comput Sci Rev* 35:100223. <https://doi.org/10.1016/j.cosrev.2019.100223>
- Noy N, Rector A (2006) PHCW: defining n -ary relations on the semantic web. <https://www.w3.org/TR/swbp-n-aryRelations/>
- Open-Link (2020) Virtuoso open-source edition. <http://vos.openlinksw.com/owiki/wiki/VOS>
- Pellens B, De Troyer O, Bille W, Kleiner F (2005a) Conceptual modeling of object behavior in a virtual environment. In: Proceedings of virtual concept 2005, Springer-Verlag, Biarritz, France, pp 93–94
- Pellens B, De Troyer O, Bille W, Kleiner F, Romero R (2005b) An ontology-driven approach for modeling behavior in virtual environments. In: Meersman R, Tari Z, Herrero P (eds) Proceedings of on the move to meaningful internet systems 2005: ontology mining and engineering and its use for virtual reality (WOM-EUVR 2005) Workshop, Springer-Verlag, Agia Napa, Cyprus, vol 3762, pp 1215–1224
- Pellens B, Kleiner F, De Troyer O (2006) Intuitively specifying object dynamics in virtual environments using vr-wise. In: Proceedings of the ACM symposium on virtual reality software and technology, ACM, New York, NY, USA, VRST '06, pp 334–337
- Pellens B, De Troyer O, Kleiner F (2008) Codepa: a conceptual design pattern approach to model behavior for x3d worlds. In: Proceedings of the 13th international symposium on 3D web technology, Los Angeles, pp 91–99
- Pellens B, Kleiner 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
- Perez-Gallardo Y, Cuadrado JLL, Crespo ÁG, de Jesús CG (2017) GEODIM: a semantic model-based system for 3D recognition of industrial scenes. In: Alor-Hernández G, Valencia-García R (eds) Current trends on knowledge-based systems. Intelligent systems reference library, vol 120. Springer, Cham, pp 137–159. https://doi.org/10.1007/978-3-319-51905-0_7
- Pouriyeh SA, Allahyari M, Liu Q, Cheng G, Arabnia HR, Qu Y, Kochut K (2018) Graph-based ontology summarization: a survey. CoRR [arXiv:1805.06051](https://arxiv.org/abs/1805.06051)
- 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. *J Biomed Semant* 6(1):36. <https://doi.org/10.1186/s13326-015-0034-0>
- SharpCrafters (2020) PostSharp. <https://www.postsharp.net/>
- 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. <https://doi.org/10.1145/3055624.3075943>
- Sikos LF (2017b) Description logics in multimedia reasoning, 1st edn. Springer Publishing Company, Incorporated, Berlin
- Spring Framework (2020) Aspect oriented programming with Spring. <https://docs.spring.io/spring/docs/2.0.x/reference/aop.html>
- Stanford University (2020) Protégé. <http://protege.stanford.edu/>
- Technologies U (2020) Unity. <http://unity.com/>
- Trellet M, Férey 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: Workshop on immersive analytics (IA). IEEE, pp 48–53
- 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
- Unity 3D (2020) ScreenCapture.CaptureScreenshot. <https://docs.unity3d.com/ScriptReference/ScreenCapture.CaptureScreenshot.html>
- 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
- W3C (2012a) OWL. <https://www.w3.org/TR/owl2-syntax/>
- W3C (2012b) OWL 2 web ontology language profiles (second edition). https://www.w3.org/TR/owl2-profiles/#Computational_Properties
- W3C (2013) SPARQL. <https://www.w3.org/TR/sparql11-query/>
- W3C (2014a) RDF. <https://www.w3.org/TR/rdf11-concepts/>
- W3C (2014b) RDFS. <https://www.w3.org/TR/rdf-schema/>
- W3C (2017) X3D. <http://www.web3d.org/getting-started-x3d>
- 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>
- Web3D Consortium (2019) X3D ontology for semantic web. <https://www.web3d.org/x3d/content/semantics/semantics.html>
- Web3D Consortium (2020) X3D semantic web. <https://www.web3d.org/working-groups/x3d-semantic-web>
- Welly 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>
- World Wide Web Consortium (2013) RDF 1.1 JSON alternate serialization (RDF/JSON). <https://www.w3.org/TR/rdf-json/>
- World Wide Web Consortium (2014) RDF 1.1 Turtle. <http://www.w3.org/TR/turtle/>
- World Wide Web Consortium (2020a) JSON-LD 1.1—a JSON-based serialization for linked data. <https://www.w3.org/TR/json-ld/>
- World Wide Web Consortium (2020b) Time ontology in OWL—W3C Candidate Recommendation 26 March 2020. <https://www.w3.org/TR/2020/CR-owl-time-20200326/>