GDAŃSK UNIVERSITY
OF TECHNOLOGY

FACULTY OF ELECTRONICS, TELECOMMUNICATIONS
AND INFORMATICS

The author of the PhD dissertation: Jakub Flotyński
Scientific discipline: computer science

DOCTORAL DISSERTATION

Title of PhD dissertation: *Semantic Modeling of Interactive 3D Content*

Title of PhD dissertation (in Polish): *Semantyczne modelowanie interaktywnych treści 3D*

Supervisor *signature*	Second supervisor *signature*
dr hab. inż. Krzysztof Walczak, prof. nadzw. UEP	<Title, degree, first name and surname>
Auxiliary supervisor *signature*	Cosupervisor *signature*
<Title, degree, first name and surname>	<Title, degree, first name and surname>

Gdańsk, 2015

Acknowledgements

I would like to thank my supervisor Professor Krzysztof Walczak for the excellent guidance and support throughout the work on the dissertation.

I would also like to express my thanks to Professor Wojciech Cellary for the valuable remarks that improved the final shape of the dissertation.

I say a word of thanks to the members of the Department of Information Technology at the Poznań University of Economics for the collaboration within a few years.

Contents

1. Introduction

Widespread use of interactive 3D technologies, such as virtual (VR) and augmented (AR) reality, has been enabled by the significant progress in hardware performance, the rapid growth in the available network bandwidth as well as the availability of versatile input-output devices. VR/AR systems become increasingly popular in various application domains, such as education, medicine, training, tourism, entertainment, social media and cultural heritage. In comparison to other types of systems, VR/AR systems are equipped with more advanced user interfaces, which offer the possibility of presenting data in the form of animated three-dimensional models with complex behavior, permit flexible interaction of users with the presented models and enable combination of the presented models with a view of the real world.

The primary element of VR/AR systems, apart from interface technologies, is interactive three-dimensional (3D) synthetic content. A number of programming libraries (e.g., OpenGL [9], Direct3D [3], Java3D [186], WebGL [69] and Away3D [2]) have been developed to enable imperative programming of 3D content with widely-used languages (e.g., C++, Java, JavaScript and ActionScript). Imperative programming of 3D content relies on the specification of subsequent steps that must be performed to achieve the desirable presentational effects. For instance, to put poles on the roofs of buildings in a 3D city model, all the buildings need to be processed in a loop. Putting the poles is performed explicitly by an algorithm that must be implemented by the content author. A different approach is employed by declarative languages, such as VRML [222], X3D [223], XML3D [16] and COLLADA [43]. Declarative programming of 3D content is based on the specification of the desirable effects that must be achieved, instead of the steps that must be accomplished to achieve the effects. For instance, a pole can be associated with a building of a particular class. Then, putting the poles on the roofs is performed implicitly, on the basis of the constraint, by a reasoning algorithm that is typically not implemented by the content author.

Interactive 3D content is characterized by diverse aspects such as geometry, structure, space, appearance, animation and behavior. The diversity of the aspects, however, makes the creation of interactive 3D content much more complicated than the creation of other types of web content such as web pages, images, audio and video. Therefore, the potential of VR/AR applications accessible on the web can be fully exploited only if efficient methods of modeling interactive 3D content are available. A number of visual environments have been developed for modeling 3D content. Advanced environments, which are intended for professional users (e.g., Blender [25] and 3ds Max [82]), offer rich capabilities of designing various content elements (such as geometry, structure, space, appearance and animation) with complex behavior described by imperative scripts. However, the complexity of such environments requires author's expertise in 3D modeling. 3D content creation may be facilitated by narrowing the domain of application and the set of available operations. Environments of this type (e.g., Ghost Productions [48], Sweet Home 3D [65] and AutoCAD Civil 3D [23]), which have been designed for domain experts (e.g., physicians, interior designers and engineers), provide tools enabling relatively fast and efficient modeling, without requiring users' extensive experience in computer graphics. This approach, however, significantly reduces the generality of the method. Regardless of the generality of the modeling method used, visual content creation with the aforementioned approaches demands user' knowledge of issues related to computer graphics, and it requires the modeling of all content components and properties to be presented.

Further simplification of 3D content creation requires separation of concerns between users with different expertise in 3D modeling, who are equipped with different modeling tools. Previous works in this area are based on parametrized templates (e.g., [228, 233, 234]) and permit content composition from reusable elements (e.g., [229,230,232]). In such approaches, different modeling tasks are separated between experts in 3D modeling and domain experts. Such approaches significantly facilitate modeling of 3D content, but they still have important limitations, which follow from their orientation on computer graphics instead of a particular application or domain.

Further progress in modeling of 3D content is possible through the use of semantic techniques [218,248]. The research on the semantic web was initiated by T. Berners-Lee and the W3C (World-Wide Web Consortium) in 2001 [85]. This research aims at the evolutionary development of the current web towards a distributed semantic database linking structured content and documents. Semantic description of web content makes it understandable for both humans and computers achieving a new quality in building web applications that can 'understand' the meaning of particular elements of content as well as their relations. This leads to much better methods of indexing, searching and analyzing content. The basis for the description of content and documents on the semantic web are ontologies. An ontology is *a specification of a conceptualization* [147]. Ontologies define such concepts as classes, properties and individuals as well as relations between them in a formal way. Applying the semantic web techniques to 3D web content is an important step towards building the 3D internet [76].

Semantic web techniques can be also used for modeling 3D content. The use of semantic techniques for modeling 3D content provides several important advantages that go beyond the current state of the art in 3D content creation. First, it enables declarative representation of 3D content [143], which is typically based on facts and rules. The order of facts and rules in declarative content representations is irrelevant, in contrast to the order of instructions in imperative content representations, e.g., loop, switch and goto instructions. In addition, facts and rules express desirable presentational effects using logical statements and implications. Therefore, their creation may be more intuitive for non-IT-specialists than programming in an imperative language. Moreover, the use of semantic techniques for 3D content representation can significantly facilitate management, analysis and processing of the content performed by web agents and search engines.

Second, semantic techniques enable conceptualization of 3D content at different levels of abstraction. Semantic concepts (e.g., classes, properties and individuals) defined in ontologies and knowledge bases may be specific to different applications and domains (medical data, virtual museum, interior design, building information models, etc.) or specific to computer graphics. Although the available programming languages and visual environments permit creation of conceptual 3D content representations at different levels of abstraction, such representations are expressed imperatively. It is not convenient for knowledge extraction and reasoning on 3D content, thereby it limits the possibilities of management, analysis and processing of content on the web.

The use of concepts related to different applications and domains, representing 3D content at conceptual levels of abstraction, facilitates content creation by users with different skills and experience—domain experts and IT-specialists. Due to the use of semantic techniques, different users can collaborate on modeling by completing substantially different tasks related to different aspects of the content and requiring different modeling tools. Although the available approaches permit separation of tasks between different users (e.g., creating different content elements or writing different parts of code), the tasks are typically specified on a common level of abstraction, thus requiring similar expertise from all involved users (e.g., the use of a common visual environment or language) and limiting the possibilities of involving experts in different domains.

Furthermore, semantic techniques enable discovery of tacit knowledge (classes and properties of individuals), which is not explicitly specified, but is the logical implication of facts and rules that have been explicitly specified in the modeled content representation. Tacit knowledge can be automatically extracted from 3D content representations, liberating developers from the requirement to specify all details of the created content [201, 202]. This aspect of modeling content is available neither for imperative languages, including the languages used in the visual environments, nor for declarative languages that have not been intended for reasoning on data, e.g., VRML and X3D.

Next, the use of semantic techniques can simplify customization of 3D content, which satisfies requirements of different content consumers—users and applications. 3D content customized by consumers may be created by different authors, using different schemes, retrieved from different sources (e.g., databases, web documents and web services) and intended for different purposes (e.g., content integration, readability and narrowing the scope of content). Existing modeling techniques, which are mostly based on specific content formats (either binary or textual) or imperative languages, are not suited to advanced analysis and use of complex formal content schemes (such as ontologies), which is necessary for customization of content based on semantics of content elements.

Finally, the use of semantic techniques enables flexible and generic content transformation to different content formats and languages enabling 3D content presentation on multiple platforms. In such an approach, once 3D content is created, it can be presented on multiple platforms in different hardware and software systems. Moreover, such an approach does not require users to install additional software, but it can leverage well-established 3D content browsers and presentation tools that may already be installed on the users' systems (e.g., Adobe Flash Player and WebGL or X3DOM-compliant web browsers). Currently, wide coverage of different hardware and software systems by 3D content presentations is usually achieved by providing separate implementations of particular 3D content browsers on individual systems, resulting in fragmentation of content and presentation technologies. Although visual environments often support different content standards, and some of them (e.g., Blender and 3ds Max) enable introduction of new standards (e.g., by implementing appropriate plug-ins), they do not enable generic, platform-independent content representation that could be efficiently transformed to different formats and languages.

The goal of this dissertation is to develop a new approach to modeling 3D content—*Semantic Modeling of Interactive 3D Content* (SEMIC). SEMIC exploits the aforementioned opportunities, which have not been exploited by the previous approaches to modeling 3D content. SEMIC enables declarative conceptual modeling of 3D content using different domain-specific ontologies and knowledge bases. SEMIC permits comprehensive 3D content representation including explicit knowledge and tacit knowledge, which is discovered while modeling. Both the explicit and the discovered knowledge are used to build generalized 3D content, which becomes the basis for on-demand content customization according to the requirements of content consumers. The generated customized 3D content is finally transformed to different content formats and languages, and can be presented on multiple platforms, including various combinations of hardware and software. The overall process of modeling 3D content with SEMIC is separated into a series of different tasks accomplished by different users, who have different skills and expertise in 3D modeling, and are equipped with different modeling tools.

The SEMIC approach consists of two interrelated elements:

1. The *Semantic 3D Content Model* (SCM) enables conceptual representation of different aspects of 3D content (geometry, structure, space, appearance, animation and behavior) at different levels of abstraction—specific to 3D content and specific to arbitrary applications or domains. 3D content representations based on SCM are declarative, as they focus on the desirable properties and relations between content elements.
2. The *Semantic 3D Content Creation Method* (SCCM) enables modeling of 3D content as a sequence of steps that are performed by different users. In different steps, different parts of SCM are used. SCCM permits knowledge discovery during modeling of content, on-demand customization of content and flexible generation of final content representations, which can be presented on multiple platforms.

The thesis of this dissertation is formulated as follows:

The SEMIC approach enables efficient creation of interactive 3D content at the conceptual level, using domain-specific ontologies.

The remainder of the dissertation is structured as follows. In Chapter 2, the state of the art in the field of 3D content creation is outlined. First, languages and libraries for programming interactive 3D content are discussed. Two groups of languages are distinguished according to the represented paradigm: imperative languages and declarative languages. Second, visual environments for modeling 3D content are presented. Visual environments offer various functions that simplify 3D content creation, such as animation and script editors, toolkits, content preview in real-time, etc. In visual environments, diverse operations on 3D content are performed manually, e.g., deployment of content objects, manipulation of meshes, drawing images for textures. Different groups of visual environments are distinguished: advanced environments, which are addressed mostly to experienced users, intuitive and user-friendly environments, which are addressed to non-IT-specialists, environments specific to particular domains (e.g., engineering, medicine and interior design), which are addressed to domain experts, and game engines, which are addressed to developers of complex 3D/VR/AR applications.

In Chapter 3, the state of the art in semantic representation and semantic modeling of interactive 3D content is presented. These two concepts are closely related. A semantic representation is a knowledge base that describes 3D content, its components (materials, textures, animations, etc.), properties and relations between components in a way that is sufficient to present the content. Semantic modeling is the process of 3D content creation that involves semantic techniques and semantic content representations. In Chapter 3, the main techniques of describing semantics of web content are presented. Next, a taxonomy of techniques for semantic representation and semantic modeling of 3D content is proposed. The taxonomy classifies the techniques in different ways, e.g., according to the addressed level of abstraction and described aspects of 3D content (geometry, structure, space, etc.). Two main groups of techniques are distinguished: schemes for semantic 3D content representation as well as methods and systems for semantic modeling of 3D content. A method is a sequence of steps in which 3D content is created, while a system is a set of interconnected software components, which differ in functions and are used to produce 3D content.

In Chapters 4, 5 and 6, the new SEMIC approach is described. Chapter 4 explains the motivations and requirements, and outlines the approach. The main advantages of SEMIC over the available approaches cover the possibility of conceptual, declarative, knowledge-based modeling of content at different levels of abstraction. SEMIC enables separation of concerns between different modeling users in content creation, on-demand customization of the created content and presentation of the content at different content presentation platforms. In Chapter 5, the SCM model is presented. SCM is a set of ontologies that enable 3D content representation at a conceptual level, specific to an application or domain, the concrete level, specific to 3D modeling, and the platform level, specific to a particular 3D content browser and a content format or language. Some ontologies describe the levels of abstraction, while other ontologies link 3D content representations specified at different levels.

In Chapter 6, the SCCM method is presented. SCCM comprises six steps, which are performed by different users and different algorithms. In SCCM, first, concrete components, specific to 3D computer graphics, are created by a modeler equipped with graphical and semantic editors. Concrete components comply with the concrete level of SCM. Second, the components are mapped to classes and properties defined in a domain-specific ontology by a technician equipped with a semantic editor. Third, 3D content is modeled at a conceptual level (determined by a domain-specific ontology of SCM) by a domain expert equipped with a semantic editor, possibly combined with a visual 3D modeling environment. Next, the conceptual content representation is linked to the concrete components through mapping, and a generalized 3D content representation is created. A generalized representation can include redundant 3D content components (to be removed in content customization), can lack some components (to be added in content customization), and can be modified in content customization. In the next step, 3D content is customized by content consumers (users and applications) by sending queries to generalized content representations. Queries specify different requirements for 3D content, such as desirable content components and properties to be presented. In the last step, the customized 3D content representation is transformed to its final counterpart, which is encoded in a particular format (obj, 3ds, blend, etc.) or language (Java, ActionScript, JavaScript, etc.) used at the platform level of SCM.

In Chapter 7, a prototype implementation of the SEMIC approach is presented. The environment, called SO-SEMIC (*Service-Oriented Semantic Modeling of Interactive 3D Content*) comprises a server and clients. The server performs computationally expensive activities, such as semantic processing of 3D content representations. The clients are used by users to design 3D content in a visual environment, according to the SEMIC approach. The clients communicate with the server, retrieve and modify content representations. The clients transform retrieved semantic content representations, which are expressed at the conceptual and concrete levels, to equivalent representations, which are expressed at the platform level, and present the representations to users. The server has been implemented using the Java language and libraries for processing ontologies, while a client has been implemented in Python as an extension of the Blender modeling environment. SO-SEMIC guides a user through the subsequent steps of SCCM in which the user leverages different parts of SCM to create different 3D content representations. The steps are enabled by different modules of SO-SEMIC.

In Chapter 8, the results of the evaluation of SEMIC are presented. Qualitative and quantitative evaluations have been conducted. The qualitative evaluation is used to assess SEMIC in terms of different aspects of representation and modeling of 3D content such as represented elements of 3D content, capabilities of conceptual, knowledge-based and multi-platform content representation, and capabilities of the use of 3D content in content repositories. In qualitative evaluation, the functionality of SEMIC has been compared with the functionality of selected representative programming languages, visual modeling environments and approaches to declarative 3D content creation. Quantitative evaluation covers: different characteristics of semantic 3D content representations (based on SCM), efficiency of 3D content creation and customization (based on SCCM) as well as the computational complexity and performance of the algorithms used in SCCM. The quantitative evaluation has been conducted using the SO-SEMIC environment.

In Chapter 9, the dissertation is concluded, the main contribution and achievements are summarized, and the possible directions of future research and development activities are indicated.

2. Creation of Interactive 3D Content

In this chapter, selected approaches to 3D content creation are discussed. The approaches are categorized into three main groups: programming languages (both imperative and declarative) and libraries, visual environments as well as solutions for multi-platform 3D content design. This review is focused on approaches supporting creation of 3D content that can be presented using web tools, as most often such approaches are currently used in modern VR/AR systems.

2.1. Languages for 3D Content Creation

A number of languages and libraries have been devised to enable imperative and declarative programming of interactive 3D content. Imperative programming of 3D content focuses on steps that must be performed to achieve the desirable presentational effects. Declarative programming of 3D content focuses on the desirable presentational effects instead of the steps that must be performed to achieve the effects. In the following subsections, the main approaches to imperative and declarative programming of 3D content are discussed.

2.1.1. Imperative Programming of 3D Content

The Open Graphics Library (OpenGL) [9] is the primary cross-platform and open source API designed for imperative programming of 3D content. OpenGL has been used in a large number of projects—computer games, simulations and visualizations—as well as multiple frameworks and libraries designed for 3D content creation. OpenGL-based applications may be launched on various operating systems—Windows, Linux, Mac OS and Android—on both desktop and mobile devices. Another API frequently used for 3D content creation and presentation, and the main competitor of OpenGL, is Direct3D [3]. Unlike OpenGL, Direct3D is not open source and it is available under a proprietary license.

There are several widely used imperative languages for creating 3D content embedded in web pages. Several libraries have been elaborated for JavaScript. The primary library for programming 3D content with JavaScript is WebGL [69], which is based on OpenGL. The use of the low-level API of WebGL may be time-consuming in large projects. Therefore, other libraries have been developed on the basis of WebGL—GLGE [49], JebGL [51] and O3D [70], which simplify programming of 3D content. 3D content based on WebGL may be presented in the majority of web browsers without installing any additional plug-ins. ActionScript is another increasingly used language for programming interactive 3D web content. Applications based on ActionScript may be presented in the Adobe Flash Player plug-in. The following libraries are available for ActionScript: Papervision3D [58], Alternativa3D [22], Away3D [2] and 3D Sandy [10]. Finally, 3D web content may be presented within Java-based applets created with the Jogl [8] and Java3D [186] libraries.

2.1.2. Declarative Programming of 3D Content

Several approaches have been devised to enable declarative programming of 3D content. The Virtual Reality Modeling Language (VRML) [222] is an open, textual language devised by the Web3D Consortium to describe static and animated 3D content in a declarative way. A VRML scene is a graph comprised of nodes reflecting different aspects of 3D content—geometry, structure, appearance, space, logic and behavior. VRML also supports linking external multimedia resources—images, audio and video. In addition to the use of specific behavioral VRML nodes (e.g., sensors and interpolators), logic and behavior of 3D objects may be described by embedded imperative ECMAScript code. Several VRML browsers are available, e.g., Cortona3D [44], BS Contact [24], FreeWRL [4] and Instant Reality [6].

The Extensible 3D (X3D) [223] is the successor to VRML. X3D introduces several functional extensions to VRML such as Humanoid Animation, NURBS and CAD geometry. Furthermore, it supports additional binary and XML-based encoding formats as well as basic means of metadata description. Depending on the desirable complexity of 3D content, different X3D profiles may be selected—interchange, interactive, immersive and full. A few X3D browsers are available, e.g., BS Contact, FreeWRL and InstantReality.

VRML and X3D enable standardized representation of 3D content on the web, which is attainable with additional plug-ins to web browsers. To enable seamless integration of X3D content with web pages, X3DOM [15] has been designed. It is an open source framework intended as a potential extension to HTML5. 3D content encoded using X3DOM can be presented without additional plug-ins in majority of the available web browsers (supporting WebGL), or with Instant Reality or Flash plug-ins—in other browsers.

XML3D [16] is another solution designed for seamless integration of 3D content into web pages. XML3D enables declarative programming of basic 3D content elements, such as groups of objects, meshes, light sources and textures. XML3D content may be presented in web browsers supporting WebGL, without installing additional plug-ins.

COLLADA [43] has been intended as a language for exchange of 3D content between different tools. The language supports programming of geometry, shaders, physics, animations and kinematics. Currently, numerous projects use COLLADA not for exchange of content but for publication of content, e.g., GLGE, Unreal Engine [14] and Unity [12].

PDF3D is another approach to publishing 3D content. It utilizes the U3D [13] file format for encoding 3D objects, and a proprietary JavaScript API for programming behavior of objects [7]. A PDF document with 3D content may be directly embedded in a web page, and presented with the Abobe Reader plug-in.

Presentation of 3D content created with the aforesaid languages is possible in the majority of web browsers. Both the imperative and declarative languages, however, have significant limitations—are inadequate for content management (indexing and searching) in online content repositories, and do not permit modeling and adaptation of content based on reasoning and domain knowledge.

2.2. Visual Environments for 3D Content Creation

A number of visual environments have been developed for manual modeling of 3D content. Advanced environments intended for professional users, such as Blender [25] (Figure 2.1) and 3ds Max [82] (Figure 2.2), offer rich capabilities of modeling various content elements—geometry, structure, appearance and animations. In addition, the environments enable implementation of complex logic and behavior of 3D content by the use of imperative programming languages (e.g., Python in Blender and MAXScript in 3ds Max). Both the environments can be extended with plug-ins developed using appropriate APIs. In contrast to 3ds Max, Blender is open source, which favors its use and modification in research projects.

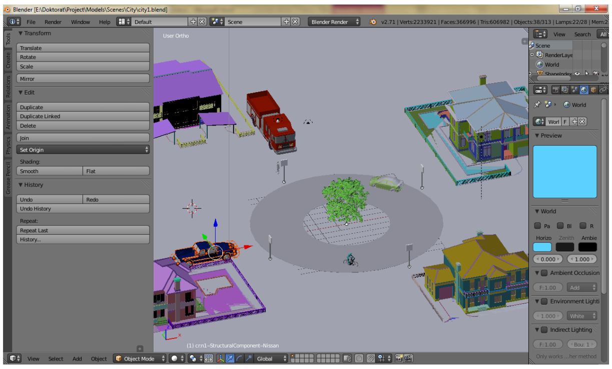


Figure 2.1: The GUI of Blender

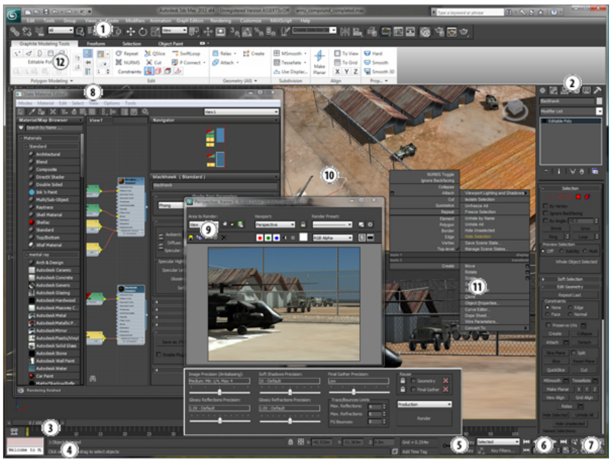


Figure 2.2: The GUI of 3ds Max (source: http://docs.autodesk.com/3DSMAX/16/ENU/3ds-Max-Help/images/interface-overview/3dsmaxUI.jpg)

User-friendly environments for modeling 3D content typically offer more limited capabilities then advanced environments, however, they are more intuitive and enable relatively quick and efficient modeling without requiring users' high technical skills. SketchUp [64] (Figure 2.3) enables creation and manipulation of 3D objects, e.g., texturing, rotating and translating objects. Furthermore, it provides a rich web repository of 3D objects that can be assembled into complex 3D scenes. 3DVIA [19] is another user-friendly environment for 3D content creation. It permits modeling of geometry, appearance, structure and animations of 3D objects.

Several user-friendly environments have been developed for modeling 3D content in specific domains, e.g., medicine, interior design and engineering. Ghost Productions [5] permits modeling of interactive 3D medical animations, e.g., surgical e-learning content. Sweet Home 3D [11] (Figure 2.4) permits modeling of interiors by drawing and editing rooms, walls, doors, windows and furniture. AutoCAD Civil 3D [1] permits creation of building information models (BIM) and modeling diverse civil constructions, such as bridges, railway tracks, pipe networks, etc.

Another group of environments for 3D content creation are game engines equipped with IDE. In comparison to environments for modeling 3D objects, game engines' IDEs permit assembly of 3D

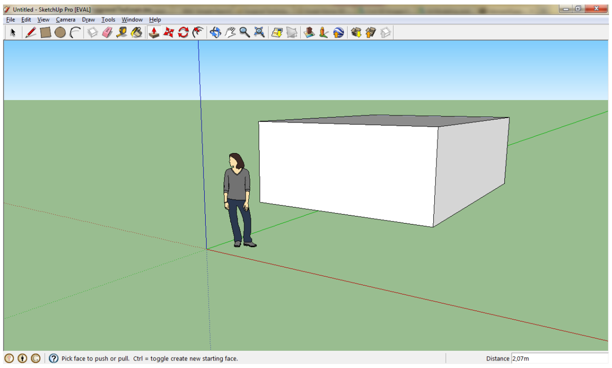

Untitled - SketchUp Pro [EVAL]
File Edit View Camera Draw Tools Window Help
Pick face to push or pull. Ctrl = toggle create new starting face.
Distance 2.07m

Figure 2.3: The GUI of SketchUp

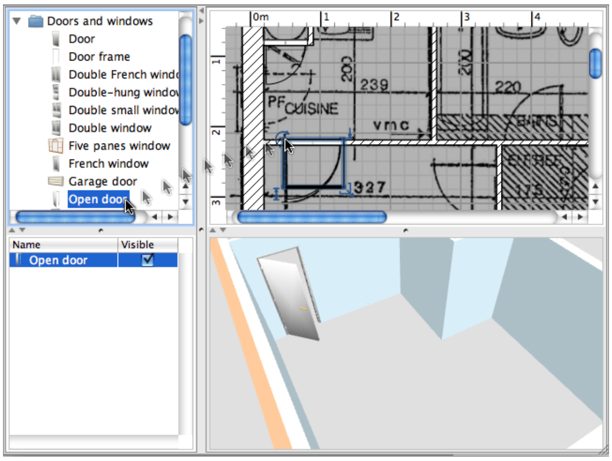
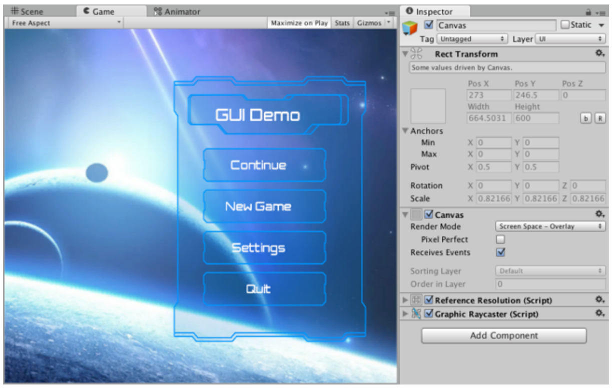


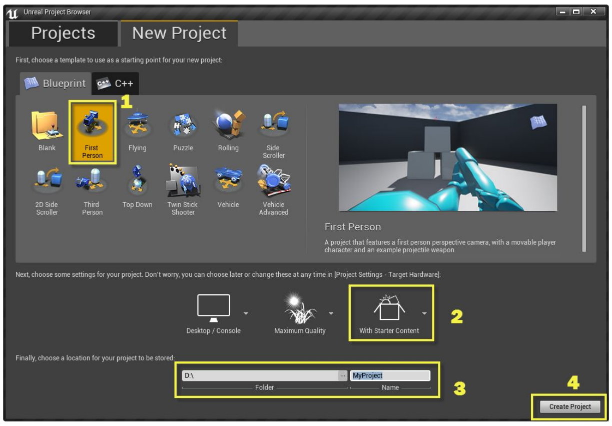
Figure 2.4: The GUI of Sweet Home 3D (source: http://www.sweethome3d.com/images/userGuide/addingFurniture.png)

objects into complex behavior-rich VR/AR applications that are collaboratively accessed by multiple users. Hence, game engines offer a number of additional functions such as modeling physics, artificial intelligence, networking, sounds, memory management and threading. Unity [12] (Figure 2.5) and Unreal Engine [14] (Figure 2.6) are widely used powerful game engines that combine visual design of 3D content with imperative programming (with C# and JavaScript in Unity and C++ in Unreal Engine). 3D content created with the engines may be presented on the web using plug-ins to web browsers—Unity Web Player and the Adobe Flash Player, respectively.



☑ Canvas □ Static
Tag Untagged Layer UI
Rect Transform
Some values driven by Canvas.
Pos X 273 Pos Y 246.5 Pos Z 0
Width 664.5031 Height 600
Anchors
Min X 0 Y 0
Max X 0 Y 0
Pivot X 0.5 Y 0.5
Rotation X 0 Y 0 Z 0
Scale X 0.82166 Y 0.82166 Z 0.82166
☑ Canvas
Render Mode Screen Space - Overlay
Pixel Perfect □
Receives Events ☑
Sorting Layer Default
Order in Layer 0
☑ Reference Resolution (Script)
☑ Graphic Raycaster (Script)

Figure 2.5: The GUI of Unity 3D (source: http://docs.unity3d.com/uploads/Main/GUI_Canvas_Screenspace_Overlay.png)



First, choose a template to use as a starting point for your new project:

A project that features a first person perspective camera, with a movable player character and an example projectile weapon.

Next, choose some settings for your project. Don't worry, you can choose later or change these at any time in [Project Settings - Target Hardware]:

Finally, choose a location for your project to be stored:

Folder: D:\

Name: MyProject

Figure 2.6: The GUI of Unreal Engine (source: https://docs.unrealengine.com/latest/images/Engine/UMG/QuickStart/1/UMGQS_1.jpg)

Although visual environments enable intuitive, manual modeling and assembly of 3D objects, they do not permit declarative knowledge-based content description and flexible content transformation to different formats and languages. Moreover, the lack of high-level conceptualization in visual environments, prevents separation of concerns in content creation between users with different skills—experts in 3D modeling and domain experts.

2.3. Multi-platform Representation of 3D Content

Several works have been devoted to 3D content representation on different hardware and software platforms. In [182], a web-based client-server system for visualization of hybrid data (georeferenced point clouds and photographs with viewpoints and camera parameters) in crisis management has been proposed. The system can be used in emergency operations and training. A specific 3D browser enables content presentation on heterogeneous devices. In [101], an approach to real-time multi-platform visualization of 3D content has been proposed. 3D content is encoded in MPEG-4 and may be rendered on platforms supporting Microsoft XNA, in particular Windows XP, Windows Vista and Xbox 360. In [75], an approach to multi-platform visualization of 2D and 3D tourism information has been presented. The system leverages a geo-multimedia database and web services, including Google Earth and Google Maps, and provides content to mobile phones, PDAs, smartphones and GPS receivers. In [212], an approach to adaptation of 3D content complexity with respect to the resources available in the target terminal has been proposed. In the approach, 3D content may be delivered with different quality. An optimization algorithm is used to ensure maximum quality for a given content rendering time. In [154], the architecture of an on-line game has been proposed. The architecture encompasses 3D game engines and a multi-platform game server. The game is accessible on PCs as well as mobile devices. In [158], an approach to integrated information spaces combining hypertext and 3D content has been proposed. The approach enables creation of dual-mode user interfaces embedding 3D scenes in hypertext, and immersing hypertextual annotations into 3D scenes, which can be presented on multiple platforms on the web. In [132], an approach to building multi-platform virtual museum exhibitions has been proposed. The system employs a client-server architecture. It supports three presentation domains: X3DOM—for presentation in web browsers without plug-ins, SWF—for presentation in browsers equipped with Adobe Flash Player plug-in, and PDF3D—for presentation in browsers equipped with Adobe Reader plug-in.

The aforementioned works cover the development of 3D content presentation platforms as well as contextual platform-dependent content adaptation. However, they do not address comprehensive and generic methods of 3D content transformation to different formats and languages.

2.4. Summary

In this chapter, selected widely-used approaches to 3D content creation have been discussed. The approaches fall into three groups: declarative and imperative languages for programming 3D content, visual environments for manual creation of 3D content as well as approaches to multi-platform 3D content representation. The analyzed solutions offer rich capabilities of creating various elements corresponding to different aspects of 3D content, such as geometry, structure, space, appearance, animations and behavior. Nevertheless, they have important limitations in the context of creation of 3D content to be used on the web. First, they do not enable declarative programming of content, which exploits inference of tacit knowledge. In general, discovery of tacit knowledge can influence the modeling process and liberates the modeling users from explicit specification of all required 3D content details. Second, the approaches provide only limited capabilities of high-level conceptual modeling of content. Although object-oriented imperative programming languages enable specification of arbitrary classes and properties, the imperative approach is not suited to automatic analysis and exploration of content (e.g., query-based), which is currently a key aspect of web applications. Moreover, the solutions do not support separation of concerns between different users with diverse skills, who can contribute to different stages of 3D content creation. Finally, the approaches do not enable generic and flexible transformation of 3D content into different content representation formats and languages, to present it in various content browsers and presentation tools. The available approaches to multi-platform content representation focus mainly on particular use cases (e.g., tourism information), tools (e.g., web browsers) and formats (e.g., MPEG-4). Existing parsers and plug-ins to 3D modeling tools (e.g., Blender) are specific to particular content formats and cannot be used for other formats. The approaches do not provide comprehensive and general solutions for generating platform-independent content representations, in particular when content must be created ad-hoc or adapted to specific contexts of use.

3. Semantic Modeling of 3D Content

This chapter provides a review of the current state of the art in semantic modeling of 3D content. The review has been structured according to a proposed taxonomy of semantic modeling approaches. The taxonomy distinguishes two main categories of approaches—*semantic schemes of 3D content*, which enable representation of different aspects of 3D content, and *semantic methods* and *tools*, which enable creation of 3D content, typically using 3D content schemes. For both the categories, several sub-categories have been distinguished with regards to the recent trends and achievements described in the literature.

In the following subsections, first, the concept of *semantic 3D content representation* that is a data model for 3D content, is explained. Second, the main techniques for representing the semantics of web content, which provide a basis for *semantic 3D content representation*, are reviewed. Third, the concept of semantic modeling of 3D content, which is the process of creating 3D content using *semantic representations*, is explained. Next, the taxonomy of semantic modeling approaches is proposed. The available approaches are classified and discussed according to the taxonomy. Finally, the review is summarized, and the main shortcomings in the available approaches as well as open issues are indicated.

3.1. Semantic Content Representation

A *semantic content representation* is a knowledge base describing the meaning of the content elements. A semantic representation conforms to an ontology that specifies a vocabulary of concepts (e.g., classes, properties and individuals) as well as relations between concepts. Ontologies are designed to be understandable and processable by both humans and computers and constitute the foundation of the semantic web [221]. In modern web-based applications, ontologies can be seen as an extension of typical metadata schemes that have been widely used, in particular, for content annotation and retrieval, e.g., [79, 242, 244], [116] and [237]. In contrast to metadata schemes, ontologies are based on formal logic and provide advanced means of expression, such as hierarchies of classes and properties, operations on sets, and chains of properties [224, 226]. The use of ontologies enables reasoning on content representations. It can lead to discovery of tacit knowledge, which has not been explicitly specified while describing the content. For instance, the link between books and their authors may enable query for the centuries when the books were written; the specification of the species of a tree may imply its typical height. The versatility and expressive power of semantic representations based on ontologies simplifies content creation, analysis and management in comparison to typical metadata schemes as well as content representation formats and languages. The usefulness of ontologies has been illustrated in multiple projects in various domains, including VR/AR, such as representing virtual environments [165], representing real and virtual cultural objects [184], representing points of interests, and facilitating navigation [163].

In the domain of virtual and augmented reality, methods of annotating, extracting, sharing and retrieving semantic content representations [120, 211], languages for designing semantic content representations as well as performance of processing semantic content representations [169] have been regarded as the main issues of modern 3D/VR/AR applications. Research works are also conducted in semantic design of behavior-rich 3D content: world representation, behavioral 3D objects and relations between objects [214].

3.2. Techniques for Representing Content Semantics

A number of techniques have been devised to create semantic representations of web resources, including multimedia, and also interactive 3D content. The approaches cover data models and languages for encoding semantic representations. The primary technique for representing the semantics of different types of web content is the Resource Description Framework (RDF) [225], which is a semantic web standard devised by W3C. RDF introduces general rules for creating semantic representations by making statements on resources. Each statement is comprised of three elements: *a subject* (a resource described by the statement), *a predicate* (a property of the subject) and *an object* (the value of the property). Every object may be either a literal value or a resource identified by an URI. RDF introduces classes, containers and lists to provide basic concepts for creating semantic representations. However, these notions are often insufficient for representing complex semantic dependencies. The RDF Schema (RDFS) [226] and the Web Ontology Language (OWL) [224] are W3C standards based on RDF. RDFS and OWL provide additional expressiveness, including hierarchies of classes and properties, constraints, property restrictions as well as operations on sets. OWL, which is based on description logic, defines a set of profiles, which differ in complexity and decidability. The possible selection of one of the available OWL profiles makes the OWL-based representations more predictable in terms of computational time and problems that are possible to solve.

The aforementioned standards enable creation of ontologies and content representations for a variety of applications and domains. However, the standards have some limitations (e.g., they do not enable calculations), which limits their expressiveness. Creation of content representations with higher expressiveness requires the use of semantic rules. The Semantic Web Rule Language (SWRL) [227] is an extension to OWL devised to represent semantic Horn-like rules [176]. SWRL offers additional expressions that enable comparison of variables, calculations as well as operations on strings, lists, date and time. Another standard which, to some extent, can be used to represent the semantics of content is the Unified Modeling Language (UML) [185] devised by OMG. UML has been intended for visual design of software. UML enables creation of diagrams based on elements representing structure, behavior and interactions.

The aforesaid solutions are not limited to represent any specific type of content. Several standards though, have been designed specifically to create semantic representations of multimedia content. The Multimedia Content Description Interface (MPEG-7) [157] provides a set of tools for creating metadata descriptions of multimedia content—Descriptors, Description Schemes, Coding Schemes and the Description Definition Language. Also some languages for 3D content representation include frameworks for describing data semantics. In VRML-based content, metadata may be embedded using `PROTO` nodes as proposed in [175]. In X3D-based content, metadata is embedded using specific nodes for encoding different data types (e.g., integer, float and string) and aggregating nodes [46]. In COLLADA-based documents, metadata attributes of particular nodes can indicate the name, id and type of the node content [42]. Specific formats of metadata are also available in some proprietary 3D content representation formats, such as FBX [47] and 3ds [17]. Some problems arising from the use of content representation frameworks and languages, such as MPEG-7 and VRML, e.g., sharing common sub-elements of a scene by multiple parental elements, have been discussed in [153].

3.3. The Concept of Semantic Modeling of 3D Content

Semantic modeling of 3D content is a process of creating 3D content representation with respect to the meaning of particular 3D content elements. Semantic modeling is a sequence of activities that are performed by humans or software. Semantic modeling covers creation, modification and processing of 3D content using semantic techniques. Due to the use of semantic techniques, semantic modeling typically emphasizes the specification of the desirable presentational results to be achieved instead of the concrete sequence of instructions that must be performed to achieve the results. Hence, semantic modeling, along with *modeling by constraints* (e.g., [174, 246]), can be regarded as a specific kind of *declarative modeling* [143], liberating developers from programming all details of the created content [201, 202]. Semantic modeling leads to the creation of a *final 3D content representation*, which may be presented using a 3D content presentation platform (e.g., browser or modeling tool). A final 3D content representation does not need to be semantic, but instead, it may be encoded in an arbitrary 3D content representation language (e.g., VRML, XML3D, JavaScript).

Semantics-based modeling paradigm for 3D media has been introduced in [126, 210]. In the works, semantic modeling has been defined as a sequence of transitions between four universes: *shape universe and knowledge domain*, *mathematical universe*, *representation universe* and *implementation universe*. In the first universe, elements of 3D content are coupled with domain knowledge to provide a *conceptual world* that is a form of 3D content representation at an arbitrarily chosen level of abstraction, e.g., virtual museum, building information models and interior design. In the mathematical universe, an appropriate *mathematical model* is used to represent the content. For instance, terrain may be modeled using bi-dimensional scalar fields. A mathematical model may be transformed into different *representation models*, e.g., solids may be represented using constructive solid geometry or boundary representation. Finally, a representation model may be transformed into different *implementation models* (data structures), e.g., a mesh is encoded using a structure with fields containing the coordinates of the vertices. Semantic modeling of 3D content is an extension of geometrical modeling, in which 3D content is typically created regardless of the meaning of its elements and without the use of semantic techniques but with specific mathematical models and content representation schemes. The role of mathematical models in geometrical modeling and a review of different content representation schemes is presented in [206].

Incorporating knowledge into 3D content representations, which occurs in semantic modeling, is one of the main aspects of building intelligent virtual environments, which may be efficiently explored with respect to the meaning of the particular elements. It exceeds the capabilities of the available 3D content representation techniques (in particular scene graphs) and requires new specific methods of mapping conceptual objects onto collections of graphical primitives [178]. A knowledge-based representation of a virtual environment can be created as an additional level of an application, enabling conceptual processing of 3D objects at a higher level of abstraction [83], e.g., actors, actions and features.

Semantic modeling, in which semantic 3D content representations are processed, is typically preceded by the creation of ontologies to which the representations conform. The creation of an ontology used in modeling may cover: the specification of a target domain, the identification of applications, gathering developers' requirements for the ontology, the identification of the key concepts of the ontology, the elicitation of competency questions for the ontology, and the initial design of the ontology [190].

Semantic modeling can be also considered as an extension of the following groups of approaches to modeling 3D content: imperative and declarative languages, visual modeling environments (discussed in Chapter 2) as well as approaches to flexible assembly and configuration of 3D content elements. In particular, the use of semantic techniques can potentially extend typical 3D content creation with knowledge-based reasoning on content representations, high-level conceptualization and platform-independence of content representations. This chapter provides an overview of the degrees to which the opportunities provided by the semantic techniques are used in the available approaches to 3D content creation.

On the basis of the available literature, semantic modeling can be regarded as a process encompassing the following four activities: *semantic reflection*, *semantic selection*, *semantic configuration* and *semantic transformation* of 3D content (Figure 3.1). Some approaches enable only single modeling activities, e.g., [100], [160], [167, 183], [119, 191], while other approaches enable several activities, e.g., [89], [74], [102], which create a content creation pipeline. Furthermore, in some approaches, all activities are based on semantic techniques, e.g., [115], [161], whereas in other approaches some activities leverage semantic techniques and other activities leverage non-semantic techniques, e.g., [91–93, 123–125, 164, 181, 196]. The activities are described below.

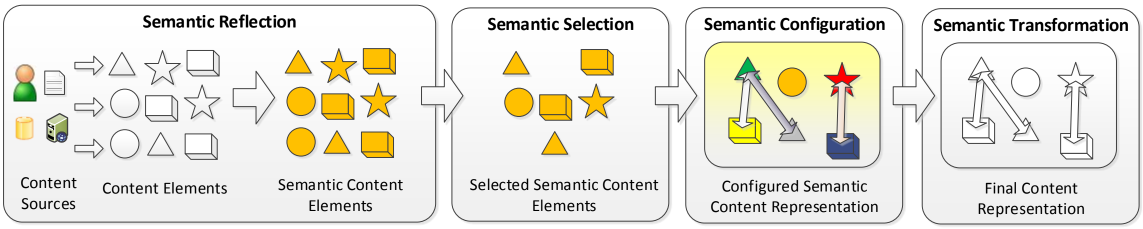

Semantic Reflection
Semantic Selection
Semantic Configuration
Semantic Transformation
Content Sources
Content Elements
Semantic Content Elements
Selected Semantic Content Elements
Configured Semantic Content Representation
Final Content Representation

Figure 3.1: The activities of semantic modeling of interactive 3D content

Semantic Reflection

Semantic reflection is the creation of semantic 3D content elements that are equivalents of non-semantic 3D content elements (shapes, coordinates, materials, animations, etc.) obtained from different content sources (e.g., files, databases [102], graphics tablets, cameras [209], web services). The created semantic elements are typically parametrized to enable configuration and generation of various final 3D content representations (e.g., [229–231, 233–236, 240]). For example, some works have been devoted to reflecting events and actions in virtual environments by semantic concepts [100], reflecting different parts of human body [183] and reflecting indoor scenes [166]. The reflected non-semantic elements are usually encoded in an arbitrary 3D content representation language (e.g., VRML, X3D, XML3D, MPEG-4), e.g., [231, 232]. If semantic reflection is followed by the other activities, semantic elements are usually encoded with a common semantic technique to preserve compatibility in the further configuration and transformation. Semantic elements may represent different aspects of 3D content, such as geometry, structure, space, appearance, animation and behavior at different—arbitrarily chosen—levels of abstraction. Further, semantic elements can modify, extend and gather the meaning of their prototype non-semantic elements, e.g., a set of meshes may be reflected as a piece of furniture; textures, transparencies and shininess may be reflected by different kinds of materials.

Semantic Selection

In semantic selection, semantic 3D content elements are chosen to create a final content representation. In the available approaches, selection usually involves only a subset of all the semantic elements that are included in a repository, e.g., [89], [124, 164], [161], [217]. Selection is performed at a level of abstraction that characterizes the elements being selected. The level may be either specific to 3D content—its geometry, structure, space, appearance and animation, e.g., [89], [102], [126], or specific to an application or domain, e.g, [74], [173,243]. Semantic selection typically precedes semantic configuration in the modeling process.

Semantic Configuration

In semantic configuration, selected semantic 3D content elements are filled with values of parameters and combined into a complex semantic representation. Similarly as in the case of selection, configuration may be performed at different levels of abstraction depending on the level characterizing the configured semantic elements—specific to 3D content, e.g. [126], [229,230,234–236,240], specific to an application or domain, e.g., [161], [197, 199], and both, e.g., [115], [160], [204].

Semantic Transformation

In semantic transformation, semantic representations of 3D content are converted into final 3D content representations, which are encoded in a 3D content representation language. Therefore, semantic transformation can be seen as an inverse semantic reflection. Semantic transformation produces a coherent content representation that may be presented, but does not necessarily consist of independent reusable content elements. Although, transformation is an important activity of semantic modeling, it is often performed without the use of semantic techniques, e.g., [98, 99, 105], [123]. In such cases, transformation is not a part of semantic modeling, however, it is undeniably a part of a broader modeling process.

In the approaches to semantic modeling that are presented in the next sections, the aforementioned activities are performed either manually (by developers) or automatically (by specific software). Automatic reflection usually requires advanced analysis (e.g., context of use) of content elements. Automatic selection and configuration are often used in contextual content adaptation, which may take into account such elements as, interaction, user preferences or profiles [231, 233] and geometrical context [77, 166], [245].

3.4. Taxonomy of Approaches to Semantic Modeling

A number of research works have been devoted to different aspects of semantic modeling of interactive 3D content. The solutions vary in several respects. In this section, a taxonomy of the available approaches to semantic modeling of 3D content is proposed. The taxonomy has been elaborated taking into account the aspects mentioned in Sections 3.1-3.3. It enables classification of the available approaches as well as identification of the main challenges and open issues in the field of semantic modeling of 3D content.

The taxonomy is depicted in Figure 3.2. Every approach is either a *scheme* of semantic content representation or a *method/tool* for semantic modeling of 3D content. The distinction between schemes and methods/tools corresponds to the distinction between data representing 3D content and logic (activities and algorithms) used to create 3D content.

— A **scheme of 3D content representation** is a description of the structure of 3D content, its elements as well as their properties and relations. For instance, an XML Schema (e.g., [115], [177]) or an ontology (e.g., [112, 113], [188, 189]) may describe classes and properties, which can be used to create 3D content representations.

— A **method of modeling 3D content** is a set of steps (e.g., [77, 166], [231, 232]) accomplished sequentially or in parallel. Some steps may by performed manually—by a human using specific hardware or software tools, while other steps may be performed automatically—by software. Completing the steps results in a final 3D content representation. However, in the intermediary steps, semantic representations are created and processed. For instance, a method of semantic reflection can consist of two steps: detecting interconnected elements of a 3D model, and analyzing the connections using a classifier [167, 183].

— A **tool for modeling 3D content** is a set of interconnected software modules that are used to interact with users, store, retrieve and process data, to enable modeling of 3D content. For instance, a modeling tool may be based on service-client architecture in which the client is implemented based on a 3D content browser, while the server may comprise several services [161]. Typically, tools use methods of modeling. However, in many works, methods are not explicitly presented, as the focus is on software modules instead of the steps performed by the modules to produce 3D content, or the steps are generally described, without sufficient details. Such works have been classified as tools in this review.

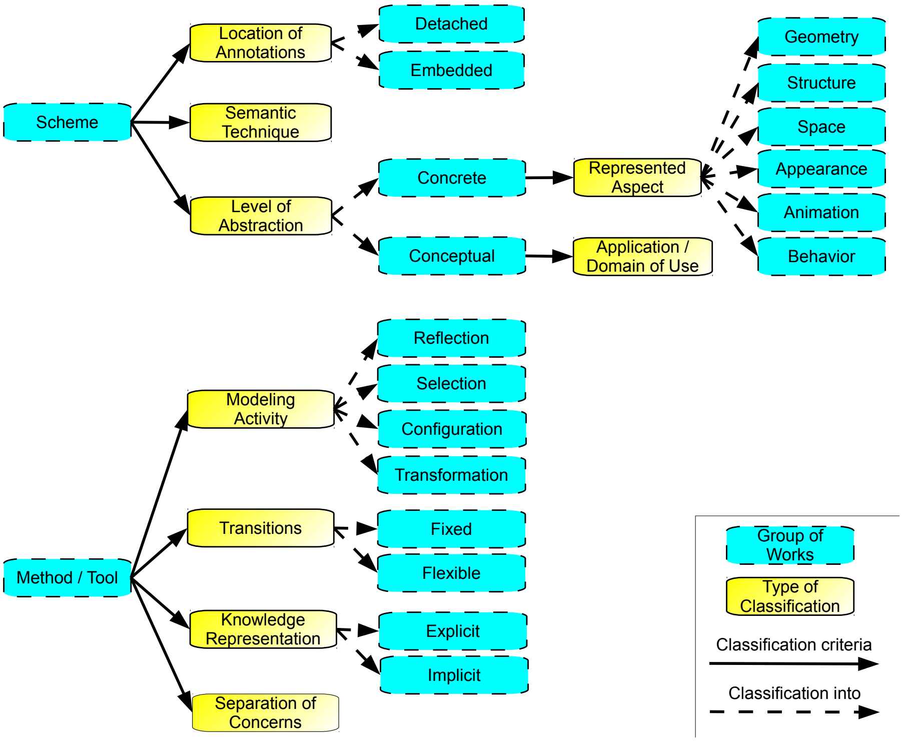


Figure 3.2: The proposed taxonomy of approaches to semantic modeling of 3D content

3.4.1. Taxonomy of Schemes

Every scheme for semantic 3D content representation may be classified in terms of the level of abstraction. On the one hand, 3D content may be represented by semantic concepts that are specific to the domain of 3D modeling, e.g., color, texture, dimensions, orientation and LODs. In this dissertation, representations that are specified at the low-level of abstraction, are referred to as *concrete 3D content representations*, e.g., [92] [73,127,210]. Such schemes are further classified according to the represented aspects of 3D content: *geometry* (e.g., coordinates, indices and shapes), *structure* (e.g., inclusion, alternatives), *space* (e.g., position, rotation and size), *appearance* (e.g., transparency, color and texture), *animation* (e.g., duration, motion trajectory, color interpolation) and behavior (e.g., activities of content elements). On the other hand, 3D content may be represented by semantic concepts that are specific to an application or domain, which is not directly related to 3D modeling, e.g., cars, furniture, exhibitions and clothes. In this dissertation, representations that are specified at a high-level of abstraction, are referred to as *conceptual 3D content representations*. Such schemes are classified in terms of the application or domain of use, e.g., virtual museum [236] [200], modeling buildings [84] [86–88] and interior design [188, 189] [74].

Furthermore, every scheme may be classified in terms of the semantic *techniques* used (e.g., semantic web standards, UML) as well as the *location of semantic annotations*. *Detached* semantic annotations form a 3D content representation that is separated from the represented final 3D content. For example, OWL knowledge bases can represent separate X3D models [160]; documents encoded in the SNIL language can represent separate 3D models [168, 171]. *Embedded* semantic annotations are included within the described document. In comparison to detached annotations, embedded annotations are more concise, reduce specification of redundant data (e.g., object IDs in both representations), and facilitate management of final representations that are inextricably linked to semantic representations. For instance, RDFa-based annotations may be embedded in X3D documents [130, 133, 135, 136]; interaction of 3D models may be represented within MIM documents [107–111].

3.4.2. Taxonomy of Methods and Tools

Methods and tools are classified in terms of the supported modeling activities, which may be one or several of: semantic reflection, semantic selection, semantic configuration and semantic transformation (cf. Section 3.3). In the methods and tools that enable several modeling activities, the activities form a content creation pipeline. For example, 3D content elements may be sequentially configured and transformed to a final representation [231, 232]. Such methods and tools are classified in terms of the supported *transitions* between activities. In the case of *fixed transitions*, 3D content is typically presented to end-users in the form obtained from the source (e.g., [91–93, 123–125, 164, 181, 196], [217]), and it is not dynamically adapted to specific requirements. In the case of *flexible transitions*, 3D content can be adapted to satisfy requirements obtained from different sources, which can be specified by different users (e.g., [89], [102], [103]). *Flexible transitions* are useful, in particular, in adaptation of commonly available 3D content that is performed on-demand, taking into account individual user preferences and the context of use (location, device, etc.).

Due to the use of knowledge representation techniques, every method and tool for semantic modeling may be classified in terms of *knowledge representation*. In *explicit knowledge representation*, the final content representation is based only on the knowledge (facts) that has been explicitly specified while modeling. For instance, in [92, 123], the mapping of domain-specific concepts to 3D content-specific concepts is explicitly specified and further used in modeling content. In *implicit knowledge representation*, the final content representation is based on both the knowledge that has been explicitly specified and the knowledge that has not been explicitly specified, but may be inferred in the reasoning process. For instance, in [245], an algorithm analyses a point cloud and derives contextual data for modeling buildings.

Addressing different modeling activities and enabling content representation at different levels of abstraction may permit separation of concerns between different modeling users who have different modeling skills and experience, and are equipped with different hardware and software tools (semantic editors, graphical editors, scanners, etc.). For instance, basic content elements may be scanned, manipulated and mapped to high-level concepts by a developer; next, the high-level concepts may be used to create a 3D content representation by a domain expert [229–231, 233, 234, 236, 244].

3.5. Schemes for Semantic 3D Content Representation

A number of projects have leveraged semantic representation of 3D content. The earliest projects in this area addressed the development of knowledge-based tools for designing 3D content [128]; the influence of semantically organized virtual spaces on the users' cognition, interpretation and interaction [104]; and modeling of historic monuments using geometrical elements combined with semantic structures [148]. More recently, in the PRISM project [59, 203], 3D models have been segmented by an algorithm and annotated by users to facilitate further analysis of the content. In the AIM@SHAPE project [21], semantic tools have enabled representation of different aspect of content such as geometry, structure, interaction and time. In the 3DSA project [18, 247], semantic approaches have been used to annotate 3D virtual museum artifacts. In this section, 3D content representation schemes are discussed according to the taxonomy proposed in Section 3.4. An overall comparison of the schemes is presented in Table 3.1. The schemes are discussed in three groups.

The schemes in the first group enable concrete representation of 3D content, which is based on concepts that are specific to 3D modeling, such as LOD, coordinates, vertices, textures, etc. Concrete content representations are typically used in methods and tools that require flexible, low-level access to 3D content elements (e.g., for querying, analyzing and retrieving) with no respect to the high-level semantics of the elements, [102], [112, 113], [121]. The aspects that are most frequently covered by schemes for concrete representations are: geometry, structure, space and appearance. Animation and behavior are the less addressed aspects, as they require more complex 3D content elements (e.g., sensors, interpolators and sequencers), [197, 199], [220], or more complex languages (e.g., rule-based), [249].

The schemes in the second—largest—group enable conceptual representation of 3D content, which is specific to a particular application or domain. Concepts used in such representations are typically not directly related to 3D modeling. The use of application- or domain-specific knowledge in content representations is an important advantage in comparison to the schemes for concrete 3D content representation. It permits 3D content creation by domain experts who are not IT-specialists [92] [231, 232].

The schemes in the third group enable multi-layered representation of 3D content, which encompasses concrete and conceptual 3D content elements. To combine both types of representations, mapping is typically used, [90, 92, 123, 125, 196]. Multi-layered representations are convenient for 3D content that needs to be represented at different levels of abstraction, e.g., primitive actions (move, turn, rotate, etc.) are combined to represent composite behaviors [125]. The majority of schemes in this group enable creation of detached annotations, which liberalizes management of both semantic and final 3D content representations, e.g., storage in different repositories.

Table 3.1: Classification of schemes for semantic 3D content representation

		Level of Abstraction							**Location of Annotations**			
			Concrete									
	Scheme	Conceptual	Geometry	Structure	Space	Appearance	Animation	Behaviour	Detached	Embedded	**Semantic Technique**	**Application / Domain of Use**
1	[126]		✓	✓	✓							industrial design, geographical modeling
2	[90, 92, 123, 125, 196]	✓	✓	✓	✓	✓	✓	✓	✓		RDF, DAML+OIL / XML	bowling game
3	[155, 168, 170–172]	✓							✓		SNIL / XML	modeling tools
4	[204]	✓	✓	✓	✓	✓		✓				modeling builldings
5	[229–231, 233, 234, 236]		✓	✓	✓	✓	✓	✓	✓		VRML, X3D, MPEG-4	virtual museum
6	[73, 127, 210]		✓	✓	✓	✓	✓					point clouds representation
7	[84]	✓									UML	modeling buildings
8	[86–88]		✓	✓	✓	✓			✓		MPEG-7 / XML	modeling buildings
9	[122]	✓										crowd simulation
10	[149, 152]		✓	✓	✓		✓		✓		MPEG-7, MPEG-21	virtual orchestra
11	[144, 149, 150]		✓	✓	✓	✓	✓	✓			RDF, RDFS, OWL / XML	virtual humans
12	[156]	✓										virtual room
13	[188, 189]	✓							✓		RDF, RDFS	interior design
14	[205]		✓		✓						OWL / XML	mobile tourist guide
15	[115]	✓	✓	✓	✓	✓	✓	✓	✓		XSD / XML	interiors adaptation
16	[160]	✓	✓	✓	✓	✓	✓		✓		RDF, RDFS, OWL, Access-Limited Logic	genome, cell description, compounds visualization
17	[183]	✓										virtual humans
18	[200]	✓								✓	RDF. X3D / XML	virtual museum
19	[80, 207]	✓							✓		RDF, RDFS, OWL / XML	virtual humans
20	[119]		✓	✓	✓				✓		RDF, RDFS, OWL	virtual furniture
21	[121]		✓	✓	✓	✓				✓	Maya Embedded Language	modeling buildings
22	[192]	✓									SMIL / XML	rigid body simulation
23	[106–111]							✓		✓	MIM / XML	virtual museum
24	[112, 113]		✓		✓		✓	✓			OWL / XML	avatars animation and interaction
25	[197, 199]							✓				
26	[94–97, 117, 118]	✓	✓	✓	✓	✓			✓		IFC / XML	BIMs
27	[159]	✓										crowd simulation
28	[177]	✓									CityGML, XSD / XML	cultural heritage
29	[162, 215, 216]	✓										game design
30	[249]	✓							✓		OWL, SWRL	order process
31	[161]	✓							✓	✓	RDFa, OWL / XML	factory simulation
32	[213]	✓									UML	virtual laboratory
33	[220]	✓	✓	✓	✓		✓				OWL	human body, product work-flow
34	[74]	✓									OWL	interior design
35	[173, 243]	✓							✓		RDF, RDFS, OWL	game design
36	[102]		✓			✓						web page design

3.5.1. Concrete 3D Content Representation

A number of works have been devoted to concrete 3D content representation. In the AIM@SHAPE project [21], a model combining final 3D content representations with corresponding concrete semantic representations has been proposed [73, 127, 210]. The model introduces four concrete levels of abstraction. The raw level covers basic content properties related to different aspects of the content such as space and appearance, e.g., dimensions and colors. The geometric level covers diverse geometrical elements, e.g., polygons, parametric surface models and structured point sets. The structural level organizes both raw and geometrical levels by enabling, e.g., multi-resolution geometry, multi-scale models and topological decompositions. Finally, the semantic level associates concrete content elements specified at the lower levels with their semantic equivalents.

In the 3DSEAM annotation model [86–88], concrete 3D content representations encompassing geometry, structure, space and appearance of the objects have been encoded as MPEG-7/XML documents and linked to representations encoded in different languages (VRML, X3D, DXF and 3ds). Similar possibilities have been offered by the Intermediate Model proposed in [126], which describes shapes and their semantics in terms of geometry, structure and space. Another scheme for concrete 3D content representation based on MPEG-7, and also on MPEG-21, has been proposed in [149, 152]. It distinguishes the following elements of 3D content: scene, semantic descriptor, digital item (an entity in a scene), geometric descriptor, shape and controller, which specifies the interactivity of digital items. The scheme may be used to represent the geometry, structure, space, appearance and animation of 3D models.

In [149, 150], an ontology for concrete representation of virtual humans has been proposed. The ontology introduces specific classes and properties. The virtual human, which is the main class, is linked to geometrical descriptors of vertices and polygons, structural descriptors of articulation levels, 3D animations of face and body, and behavior controllers (animation algorithms). The extension of virtual humans with semantically represented emotional body expressions is possible by applying the ontology proposed in [144]. The ontology is built upon the Whissel's wheel activation-evaluation space [241]. It includes concepts combining passive/active and negative/positive adjectives related to human emotions, e.g., despairing (very passive and very negative), furious, terrified and disgusted (very active and very negative), serene (very passive and very positive), exhilarated, delighted and blissful (very active and very positive).

In [112, 113], ontologies for representing multi-user virtual environments and avatars (Figure 3.3) at the concrete level have been described. The ontologies focus on the geometry, space, animation and behavior of 3D content. Virtual environments may be represented using a number of concepts that are semantic counterparts to concepts incorporated in widely-used 3D content representation languages (e.g., VRML and X3D). World objects, which are the main entities of the content, are described by such elements as translation, rotation and scale. Avatars are described by names, statuses and UIs, while their behavior is described by code bases.

In [197, 199], a graphical notation for modeling behavior of 3D content has been proposed. The notation includes diagrams for behavior definition and behavior invocation (Figure 3.4). Diagrams are configured according to design patterns that specify solutions to frequent design problems, e.g., suspend-resume pattern, path movement pattern and customized capture evade pattern.

The ontology proposed in [119, 191] permits concrete representation of non-manifold 3D models, e.g., a spider-web on a window, an umbrella with wires, a cone touching a plane at a single point. The ontology includes such properties as the number of vertices, number of non-manifold vertices, number

Figure 3.3: A virtual environment with an avatar represented using ontologies. Source: [112]. Courtesy of Y. Chu and T. Li

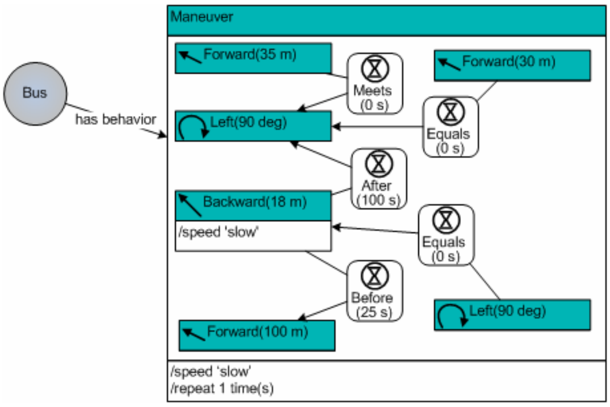


Figure 3.4: A behavior definition diagram compatible with a semantic graphical notation. Source: [197]. Courtesy of B. Pellens, O. De Troyer and F. Kleinermann

of edges, number of non-manifold edges, number of connected elements, etc. The Common Shape Ontology [220], which also stresses representation of shapes, focuses on geometry, structure, shape and animation of 3D content by providing such concepts as manifold and non-manifold shapes, point sets, hierarchically structured groups of objects, position, orientation and key frame animations. In addition, the ontology enables basic description of final content representations, e.g., creator, institution and file information.

The scheme of concrete representation of historic buildings, proposed in [121] introduces such concepts as dominant surface, transitions, repetition (which permits to organize elements according to symmetry or rhythm) and mouldings. Semantic representations are embedded in the content using nodes of the Maya Embedded Language. A 3D model representing a fragment of a historic building, which can be described using the scheme, is presented in Figure 3.5.

In the Flex-VR approach [229–231,233,234,236], a wide range of aspects characterizing 3D content (geometry, structure, space, appearance, animation and behavior) may be represented using generalized and parametrized content objects (VR-Beans). VR-Beans are basic, configurable content elements represented by scripts encoded in VRML, X3D or MPEG-4. VR-Beans may be linked to various media elements, such as texts, images, audio and video. The behavior of VR-Beans is described using the declarative VR-BML language [229]. Content representations in Flex-VR may be hierarchically organized into presentation spaces (containers of objects) and presentation domains (corresponding to different contexts of content presentation).

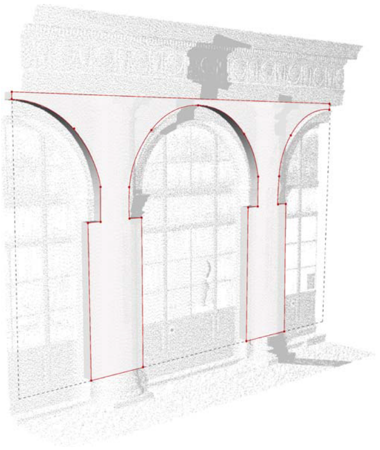


Figure 3.5: A reconstruction of a dominant surface, which can be described using a semantic scheme. Source: [121]. Courtesy of L. De Luca, P. Véron and M. Florenzano

3.5.2. Conceptual 3D Content Representation

A number of the analyzed schemes enable conceptual 3D content representation. In the approach presented in [90, 92, 123, 125, 196], conceptual 3D content representations may be designed on the basis of different domain-specific ontologies. The proposed ontologies based on DAML+OIL [45] and OWL are used to determine different aspects of the content. The ontologies mainly emphasize spatial and structural dependencies between objects (position, orientation and connection) as well as primitive content behavior (e.g., move, turn and resize).

The Knowledge Representation Layer (KRL) presented in [155, 168, 170, 171] is used to incorporate conceptual semantic representations into virtual environments. It specifies entities representing simulated objects (e.g., rods), attributes of entities (e.g., color), concepts that are categories of objects (e.g., a class of rods), and relations that are n-ary predicates of entities, attributes and concepts (e.g., is part of). KRL has been implemented using the Semantic Net Interchange Language (SNIL), which is an XML dialect.

The ontology discussed in [156] enables conceptual 3D content representation based on classes and properties related to interior design, e.g., table, lamp, teapot and floor. The properties specified in the ontology are mostly related to space and behavior, e.g., relative location and degrees of freedom. Other ontologies with concepts pertaining to interior design have been used in [188, 189] and [74].

In [183], concepts representing different parts of human body (legs, arms, fingers, neck, etc.) have been proposed. Another ontology for conceptual representation of human body has been proposed in [80, 207]. The ontology specifies a hierarchy of classes corresponding to different parts of body (head, arm, leg, foot, etc.) as well as properties describing dependencies between different classes (e.g., *is a* and *part of*). The set of conceptual semantic attributes proposed in [102] are used to represent the strengths of different features of 3D models, e.g., 3D models of animals that are scary to varying degrees differ in the appearance of the snout.

A model combining conceptual 3D content representations (encoded in OWL) with X3D content representations has been proposed in [200]. Three concepts are used within the ontologies to specify the relative positions of content objects: contained, shared and bounded. The concepts are mapped to particular nodes in the modeled X3D representations using nodes intended for metadata description in X3D (MetadataSet and MetadataString). X3D metadata nodes enable semantic representations that are directly embedded in the final content representations. In [192], a general idea of a content model incorporating a scene graph (representing objects with spatial relations), a time graph (representing real spaces with temporal relations) and an ontology (representing domain-specific knowledge with semantic relations) has been proposed. In the model, ontologies are used to couple conceptual semantic elements with their synthetic- and real-world counterparts.

In [173, 243], 3D content is represented using an actor model to enable real-time interactive simulation. Actors are the main elements of the created content, which manage entities—collections of semantic properties describing different content objects. The communication between actors is based on events and shared variables. OWL-based ontologies are used to provide a knowledge representation layer for conceptual representation of content elements.

In [161], conceptual semantic representations may be either detached or embedded in the final representations. Semantic representations are encoded in XML using the RDFa and OWL standards. Semantic representations used in the tool are linked to content objects encoded in XML3D documents. In [133, 135], a method of creating semantic content representations that are directly embedded in the final content representations has been proposed. The method uses metadata nodes and attributes that are available in X3D. Embedded semantic representations can be extracted from the final representations and combined into detached semantic representations depending on different aspects, e.g., the structure, types and roles of the described content elements [130]. The method may be used to embed microformat- and microdata-based semantic representations into final representations [136]. The schemes contain different properties related to structure, space, logic, time and behavior, such as built-in media elements, dimensions, packages required, interactivity and animations.

The ontology presented in [122] enables conceptual representation of crowd simulations. The ontology includes such classes as agent, agent's profile (e.g., employed adult, unemployed adult and child), place (e.g., house, church and work) and schedule related to places (e.g., opening and closing time). The properties of semantic individuals include, names, identifiers, dimensions, destinations, etc. Another conceptual representation of crowd has been proposed in [159]. The representation includes multiple layers. Data semantics is described at the semantic level, which is connected to the geometry level, in three respects: the structure of the 3D scene (e.g., inclusion of floors, stairs and corridors), its topology (e.g., connective relations between separate objects, such as stairs and the floor) and its height coordinates (as the modeled environment may include several levels)—Figure 3.6.

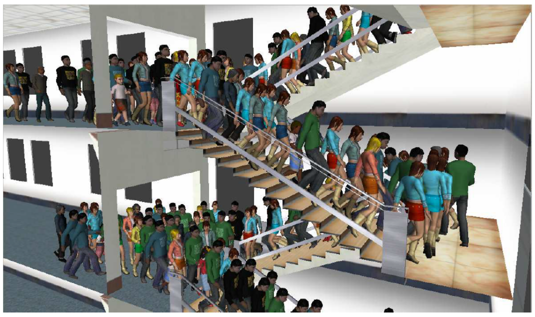


Figure 3.6: Crowd in a multi-layered virtual environment, which is represented using ontologies. Source: [159]. Courtesy of H. Jiang, W. Xu, T. Mao, C. Li, S. Xia and Z. Wang

In [177], a scheme for conceptual 3D representation of buildings has been elaborated. The scheme has been used in the 3D reconstruction of Forte San Giorgio on Capraia Island (Figure 3.7). The scheme is encoded in CityGML [40] and XSD [71]. It represents different classes of parts of buildings, functions and usage. Moreover, it specifies properties for content description, e.g., roof type, number of storeys, year of construction and demolition. Also the ontology proposed in [205] is used to represent 3D models at a high-level of abstraction, e.g., historic buildings. The ontology includes concepts describing geometrical models with spatial properties.

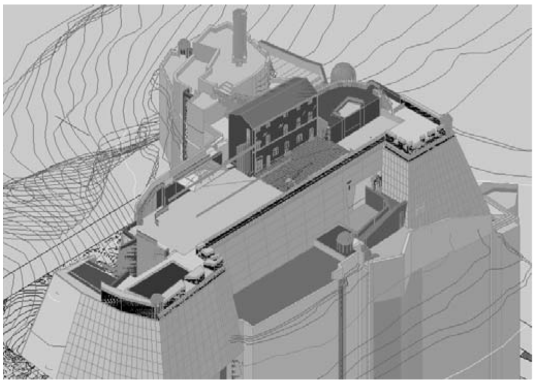
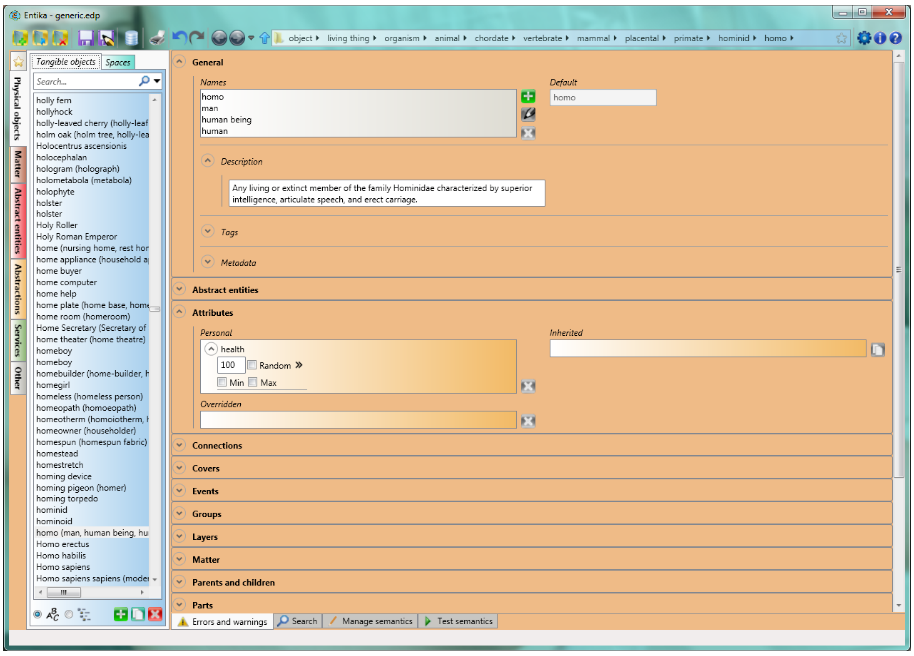


Figure 3.7: A semantically represented and described 3D reconstruction of the Forte San Giorgio. Source: [177]. Courtesy of M. Lorenzini

The scheme for conceptual content representation proposed in [162,215,216] is used in game design. It represents independent content elements as entities. Entities are divided into two groups: abstract entities (without visual representation) and physical entities (with visual representation) which, in turn, are divided into spaces and tangible objects (made of matter). Basic features of entities are described by attributes, e.g., genre and number of pages. In addition, objects may have predicates (adjectives) assigned, e.g., a comfortable chair, an antique closet. Objects may invoke services. For a service, an event and an action are semantically represented. An event has an effect, e.g., a reaction, a change and a transformation. Semantic content representations may be created using the Entika semantic editor (Figure 3.8).



holly fern
hollyhock
holly-leaved cherry (holly-leaf
holm oak (holm tree, holly-lea
Holocentrus ascensionis
holocephalan
hologram (holograph)
holometabola (metabola)
holophyte
holster
holster
Holy Roller
Holy Roman Emperor
home (nursing home, rest hor
home appliance (household a
home buyer
home computer
home help
home plate (home base, home
home room (homeroom)
Home Secretary (Secretary of
home theater (home theatre)
homeboy
homeboy
homebuilder (home-builder, h
homegirl
homeless (homeless person)
homeopath (homoeopath)
homeotherm (homoiotherm, h
homeowner (householder)
homespun (homespun fabric)
homestead
homestretch
homing device
homing pigeon (homer)
homing torpedo
hominid
hominoid
homo (man, human being, hu
Homo erectus
Homo habilis
Homo sapiens
Homo sapiens sapiens (mode

Names
homo
man
human being
human

Default
homo

Description
Any living or extinct member of the family Hominidae characterized by superior intelligence, articulate speech, and erect carriage.

Tags

Metadata

Personal
health
100 □ Random »
□ Min □ Max

Inherited

Overridden

Figure 3.8: The user interface of the Entika semantic editor simplifying creation of semantic content representations. Source: [162]. Courtesy of J. Kessing, T. Tutenel and R. Bidarra

In [249], an OWL- and SWRL-based ontology for modeling features of objects in different domains has been proposed. The ontology specifies different types of features, compositions of features (conjunction and alternative), attributes of features (variables associated with features), relations between features (mandatory or optional) and constraints on features (e.g., excludes, implies, extends, equal, greater and lesser). Furthermore, the created semantic 3D content representations may be verified in terms of consistency, e.g., an object that is required by another object cannot exclude the use of that requiring object.

In [213], a scheme for creating conceptual 3D content representations based on UML diagrams combined with constraints encoded in the Object Constraint Language [55] has been proposed. The concepts represented using UML correspond to different equipment of a physics laboratory, e.g., lenses, mirrors and lights. Constraints are built and linked to UML-based elements in an object-oriented fashion—by referring to the identifiers of the target objects. For instance, the relative position of the zone of a transmitter, a lens and a mirror may be imposed by a constraint (Figure 3.9).

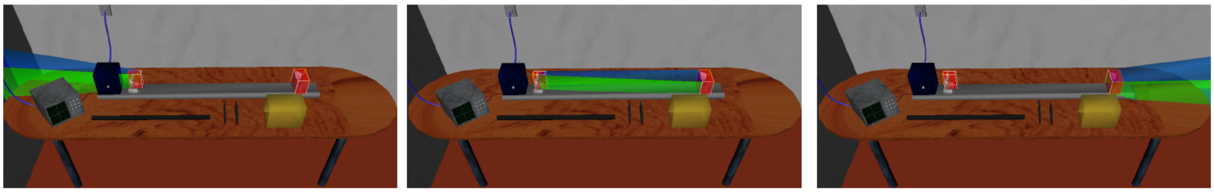


Figure 3.9: The visualization of different semantically specified subspaces between a transmitter, a lens and a mirror: the zone before the lens and the mirror, the zone between the lens and the mirror, and the zone after the lens and the mirror. Source: [213]. Courtesy of T. Trinh, R. Querrec, P. De Loor and P. Chevaillier

3.5.3. Multi-layered 3D Content Representation

A few of the analyzed schemes enable multi-layered semantic representation of 3D models of buildings. In the Virtual Design Environment (VDE) presented in [204], conceptual content representation is used to represent content elements that are specific to the domain of modeling buildings, e.g., walls, doors and windows. Concrete content representations, which are combined with conceptual representations, represent various aspects of the content, such as geometry, structure, space, appearance and behavior. In [84], an UML-based scheme for conceptual representation of building models has been proposed. The scheme includes different classes (e.g., wall, rooms, passages and storeys) and properties (e.g., volume, surface, curve and LOD). Furthermore, mapping of the scheme to the Industry Foundation Classes (IFC) has been presented. Another multi-layered scheme for representing buildings has been proposed in [94–97, 117, 118]. The scheme is based on the Industry Foundation Classes IFC [50]. The elements of the scheme describe geometry (e.g., bounding boxes, solid models, face sets, walls, windows and columns), structure and space (e.g., aggregated objects and storeys).

A general-purpose scheme for 3D content representation has been proposed in [115]. Both conceptual and concrete 3D content representations are encoded using the XML Schema and linked to 3D content using XML attributes. 3D content is represented with the VRML and X3D languages. In [160], an ontology providing concrete elements that are equivalent to elements specified in X3D (e.g., group, transform, light, color, normal index and crease angle) has been proposed. In addition, a set of semantic properties have been introduced to enable mapping of concrete elements to conceptual elements, e.g., represents, equivalence, similarity and disjointness.

3.6. Methods and Tools for Semantic Modeling of 3D Content

In this section, methods and tools of semantic modeling of 3D content are discussed according to the proposed taxonomy. An overall comparison of the methods and tools is presented in Table 3.2. The majority of the works cover tools, which are sets of interconnected software modules developed for 3D content creation. A smaller group of works cover methods, which are generic sequences of steps leading to the creation of 3D content.

The considered methods and tools usually support only subsets of the semantic modeling activities introduced in Section 3.3. The largest group encompasses works supporting content selection and configuration, e.g., [126], [204], [250]. Another group comprises works that support semantic reflection but do not support the other modeling activities, e.g., [167, 183], [80, 81, 207], [209].

About one-third of the works uses reasoning to enable discovery of tacit knowledge, e.g., [74], [100], [160]. Such solutions are typically more flexible and require less effort in modeling than the works that only make use of the explicit knowledge, e.g., [194], [204], [250]. Knowledge discovery permits more comprehensive 3D content representation, e.g., indirect links between conceptual and concrete representations.

A few works explicitly divide the modeling process into separate activities that may be accomplished by different users. Separation of concerns is enabled more often by tools then by methods, e.g., [98, 99, 105]. In tools, different modeling activities are typically related to different software modules which, in turn, can be used by users with different skills in 3D modeling.

The majority of the methods and tools do not enable *flexible transitions* between different modeling activities. The methods and tools enable content creation based on data obtained from single sources. A relatively small group of works enrich primary semantic content representations (specified in content repositories) with queries dynamically specified by content users [89] or rules specified within separate content adaptation profiles [115].

Table 3.2: Classification of methods and tools for semantic modeling of 3D content

	Method / Tool	**Modeling Activity**				**Separation of Concerns**	**Knowledge Representation**		**Transitions**		**Application / Domain of Use**
		Reflection	Selection	Configuration	Transformation		Explicit	Implicit	Fixed	Flexible	
1	[126]		✓	✓				✓	✓		industrial design, modeling spatial data
2	[145]			✓				✓	✗		interior design
3	[100]	✓						✓	✗		avatars' behavior
4	[98, 99, 105]		✓	✓		✓	✓		✓		aircraft simulation, cultural heritage,training firefighters
5	[204]		✓	✓			✓		✓		modeling buildings
6	[91–93, 123–125, 164, 181, 196]		✓	✓		✓	✓		✓		bowling game, 3D shop, search engine
7	[229–231, 233–236, 240, 244]	✓	✓	✓		✓	✓		✓		virtual museum
8	[149, 151]				✓		✓		✗		virtual characters
9	[114, 195, 197–199, 219]			✓			✓		✗		RPG game, driving simulator
10	[115]		✓	✓	✓		✓			✓	interiors adaptation
11	[160]			✓				✓	✗		genome, cell description, compounds visualization
12	[167, 183]	✓						✓	✗		virtual humans, shape segmentation
13	[80, 81, 207]	✓						✓	✗		virtual humans
14	[89]		✓	✓			✓			✓	modeling buildings
15	[119, 191]	✓					✓		✗		virtual furniture
16	[179, 180]			✓				✓	✗		fluid simulation
17	[161]		✓	✓			✓		✓		factory simulation
18	[245]	✓						✓	✗		modeling buildings
19	[74]	✓	✓	✓		✓		✓	✓		interior design
20	[103]		✓	✓				✓		✓	assembling shapes
21	[129, 173, 243]		✓	✓			✓		✓		game design
22	[77, 166]	✓							✗		interior design
23	[217]		✓	✓		✓	✓		✓		modeling buildings
24	[209]	✓							✗		interior design
25	[102]	✓	✓	✓				✓		✓	modeling shapes
26	[250]		✓	✓			✓		✓		modeling furniture
27	[238]		✓	✓				✓		✓	virtual museum
28	[194]	✓					✓		✗		simulation

3.6.1. Methods of Semantic Modeling

The methods of modeling 3D content have been divided into three groups in terms of input data. The first group includes methods designed to create content representations based on 3D point clouds and images. The second group includes methods that use 3D meshes for modeling 3D content. The third group includes methods designed for modeling content with reusable 3D content elements. The methods are presented with focus on particular steps that need to be performed to produce final 3D content representations.

Semantic Modeling with Point Clouds and Images

Several works have addressed semantic representation of 3D point clouds (typically obtained from laser scanners) and objects detected in images (typically obtained from RGB-D cameras). The works are related to modeling of interiors and buildings. In [77, 166], a method of semantic annotation of indoor scenes has been proposed (Figure 3.10). The view of a scene consists of multiple images obtained from an RGB-D Kinect camera mounted on a robot. The following characteristics are detected in the view: visual appearance (colors, textures, gradients, intensities, etc.), local shape and geometry (a table is horizontal, a keyboard is uneven, etc.) and geometrical context (e.g., a monitor is usually on top of a table). The view is segmented and semantic concepts are assigned to the segments using a maximum-margin classifier, which maximizes a discriminant function calculated on the characteristics.

Annotating 3D scenes obtained from an RGB-D camera has also been addressed in [209] for semantic modeling of 3D indoor scenes. The views are segmented, and the conditional random field algorithm assigns conceptual annotations to the recognized elements. After the semantic reflection, 3D models corresponding to the annotations are retrieved from a database and combined into a final representation.

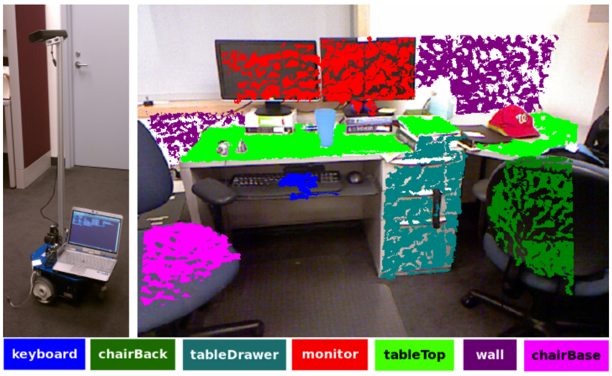


Figure 3.10: (Left) Cornell's Blue robot mounted with a camera (Microsoft Kinect). (Right) Predicted semantic representations of the scene elements. Source: [77]. Courtesy of A. Anand, H. S. Koppula, T. Joachims and A. Saxena

The method proposed in [245] uses contextual data for creating semantic 3D models of buildings on the basis of scans. The method processes point clouds obtained from a laser scanner to detect planar patches. The patches are classified using the conditional random field algorithm. The algorithm analyses the relations between the neighbouring patches (contextual information) and assigns semantic elements that are equivalents of the analyzed geometrical elements, e.g., walls, floors and ceilings. The resulting model is a semantic reflection of the input point cloud.

Another method of creating semantic equivalents of 3D point clouds of buildings has been proposed in [74]. In the first step of the method, an input point cloud is analyzed to discover planar patches, their properties (e.g., locations) and relations. In the next step, an OWL reasoner processes a domain-specific ontology including conceptual elements that potentially match the analyzed patches. Matching elements are selected and configured to build a conceptual representation. The representation is a semantic reflection of the input point cloud, which complies to the ontology. Finally, the representation is verified against the input scan using the iterative closest point algorithm. In addition, the method enables the separation of concerns in modeling, as the ontologies may be designed by a developer, while 3D scanning and using the algorithm may be performed by domain experts.

Semantic Modeling with 3D Meshes

A few works have addressed semantic reflection of various properties of 3D meshes. In some works, semantic reflection is performed after content segmentation, in which different elements of the content are distinguished on the basis of their properties (geometry, colors, relative locations, etc.). In such cases, the semantics of content elements is usually determined by their properties and the context of use within the content, which is indicated by dependencies between the elements.

The method proposed in [167, 183] detects connections between elements of a 3D model and generates a graph representing the analyzed content (Figure 3.11). The similarity between the walks of different elements in the graph indicates similar semantics of the elements which, in turn, determines their functionality. The functionality of content elements is analyzed using unsupervised classifiers, e.g., the randomized cut algorithm, which assigns the most appropriate annotations to particular content elements.

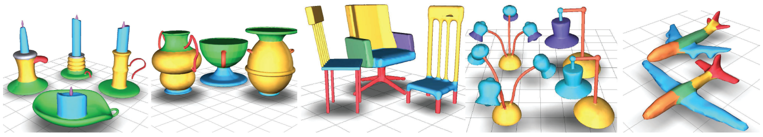


Figure 3.11: Part-wise semantic correspondences between 3D shapes. Source: [167]. Courtesy of H. Laga, M. Mortara and M. Spagnuolo

In [80, 81, 207], after automatic segmentation of 3D content, the distinguished elements are semantically annotated. Annotation may be performed automatically—by software considering topological relations between features (e.g., adjacency and overlapping) and geometric aspects of content features (e.g., orientation and size). Manual annotation may be performed by a user equipped with a graphical tool. The authors discuss both keyword- (not formalized) and ontology-based (formalized) approaches to annotating, and they present an example using an ontology of human body.

The method proposed in [119, 191] enables reflection of non-manifold 3D models using concrete properties. The method identifies properties related to the following aspects of the models: non-manifold singularities (e.g., isolated points and curves), one-dimensional parts, connected elements and maximal connected elements. Once identified, the properties are mapped to a common shape ontology and form a concrete representation of the analyzed model. In addition, the method offers flexible transitions between modeling activities, since the analysis may be supported by a user who manually segments the model.

Semantic Modeling with Reusable Elements

Several works have addressed modeling of 3D content based on semantic reusable 3D content elements. In [145], a method of automatic configuration of interior scenes based on constraints has been proposed. In the method, semantic scene representations include conceptual individuals and properties, e.g., table, floor and stands-on, as well as constraints on individuals and properties, e.g., a scene is valid if no objects that are not connected by a constraint are colliding. The final scene representation is inferred using an algorithm that operates on the semantic scene graph and dynamically assembles scene objects according to their constraints.

The method proposed in [91–93, 123–125, 164, 181, 196] enables ontology-based modeling of 3D content at a conceptual level. Content selection and configuration are performed with domain-specific ontologies, which are mapped to a 3D content-specific ontology. Final content representations are generated by the method in a way that is specific to a particular content representation language. The separation of concerns in modeling is possible, since conceptual content representations may be created by domain experts according to the domain-specific ontologies, while the 3D content-specific ontologies may be used by a developer who is an expert in semantic modeling.

The method of modeling 3D content proposed in the Flex-VR approach [229–231, 233, 234, 236] encompasses content reflection as well as selection and configuration of content elements. Content configuration may be manual (performed by a user) or automatic (performed by software). Commonly occurring problems are solved using content design patterns. Activities on generalized content objects do not require advanced technical skills in 3D modeling. Therefore, separation of concerns is possible—content objects may be prepared by developers and further used by domain experts building 3D scenes at a domain-specific level. An example of a virtual museum scene modeled by a domain expert using Flex-VR is presented in Figure 3.12.

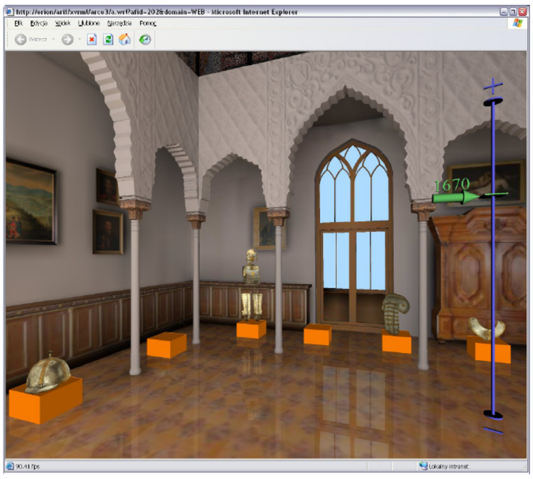

http://orion/arif/xvrml/arco3/a.wrl?afid=202&domain=WEB - Microsoft Internet Explorer

Figure 3.12: A virtual museum scene designed with Flex-VR. Source: [231]

The method proposed in [160] enables 3D content configuration. It uses ontologies encoded in OWL as schemes for conceptual and concrete content representation. Concrete content representations are created with concepts that are equivalents of concepts specified in X3D. Semantic rules, which are encoded in the Access-Limited Logic language, permit flexible transitions between the modeling activities. For example, to every object that is an instance of the shape class, assign a preferred material. Both OWL statements and rules are processed by a reasoner to discover tacit knowledge, which influences the final content representation.

In [103], a method of probabilistic reasoning for assembly-based modeling of 3D content has been proposed. The method is based on a Bayesian network, and it has two stages. In the first stage, preprocessing of a training set of segmented and annotated 3D models is performed to distinguish different categories of models with similar geometrical features. In the second stage, probabilistic reasoning is performed in real time, while modeling 3D content. Elements from the most appropriate categories are selected and configured with the current form of the content when chosen by the modeling user. For instance, after the selection of a fuselage, wings are proposed by the method to be selected (as they are probable in this case), whereas no wings are proposed when the modeled plane already has two wings (as the presence of more then two wings is unlikely in this case). Examples of models assembled using the method are presented in Figure 3.13.

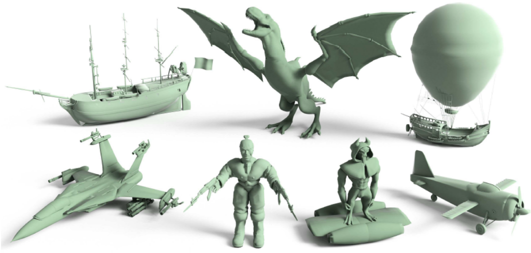


Figure 3.13: Examples of 3D models semantically assembled using probabilistic reasoning. Source: [103]. Courtesy of S. Chaudhuri, E. Kalogerakis, L. Guibas and V. Koltun

In the Smart Variations approach [250], 3D models are automatically generated on the basis of triplets of other 3D models (Figure 3.14). Selection and configuration of basic shapes are based on geometrical arrangements among symmetrically related substructures that determine the semantics of the shapes and their parts.

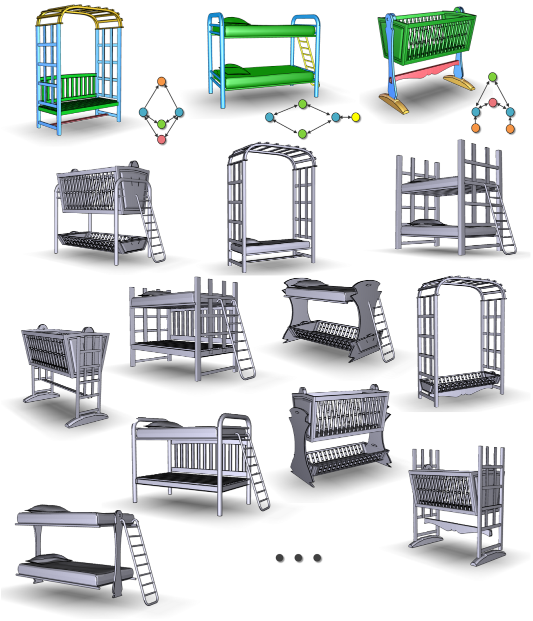


Figure 3.14: Numerous 3D models generated using symmetric functional arrangements detected in three input models. Source: [250]. Courtesy of Y. Zheng, D. Cohen-Or and N. J. Mitra

3.6.2. Tools for Semantic Modeling

The tools for semantic modeling of 3D content have been grouped in terms of the type of the content produced. Three groups of tools have been differentiated. The first group includes tools designed to build interactive 3D simulations. The second group includes tools designed for modeling buildings and spatial data. The third group includes tools designed to create adaptable 3D content. The tools are presented with the focus on particular modules that are used to produce final 3D content representations.

Semantic Modeling of 3D Simulation

Several works have addressed semantic modeling of interactive 3D simulations. In the REALISM Artificial Intelligence Virtual Environment (RAIVE) tool [100], high-level (conceptual) representation of events and actions is used for modeling the behavior of avatars. The semantics of behavior is reflected by mappings between natural language constructs and sets of basic physical events, such as, moving, changing direction, picking up an object, etc. The interpretation of a command invokes a sequence of actions, e.g., interactions with the environment. Mappings are created using a finite-state transition network (FSTN).

In [98, 99, 105], the MultiAgent System for Collaborative, Adaptive & Realistic Environments for Training (MASCARET) has been presented. The selection and configuration of content elements are performed by users equipped with a 3D modeling tool (Figure 3.15). The tool refers to UML documents that specify semantic representations of content elements. The tool permits transformation of UML-based scenes to final 3D scenes. The separation of concerns is possible, as the preparation of UML-based content models is performed by a UML developer, while the modeling activities are performed by a specialist in 3D modeling. Finally, the UML-based content representation is automatically transformed and launched on a selected content presentation platform.

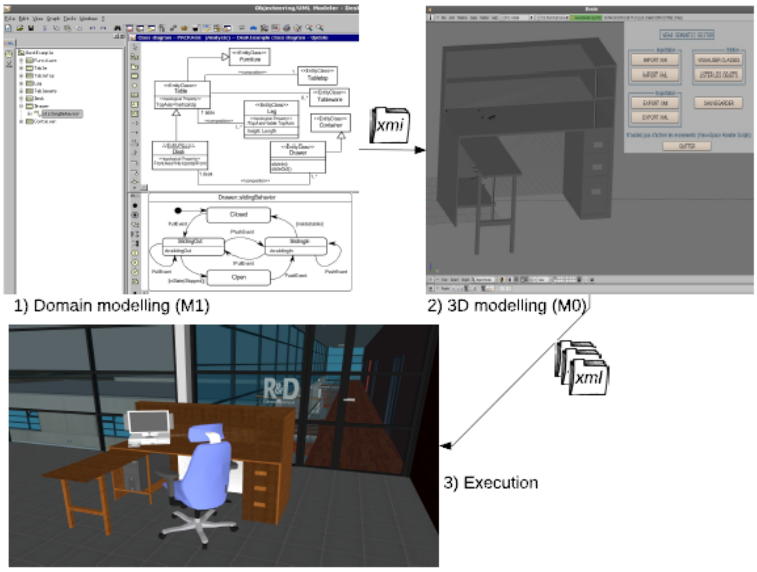


Figure 3.15: Semantic modeling 3D content with MASCARET. Source: [105]. Courtesy of P. Chevaillier, T. Trinh, M. Barange, P. D. Loor, F. Devillers, J. Soler and R. Querrec

In [179, 180], a tool for creating knowledge-based simulations has been described. The tool uses conceptual semantic representations of events and actions (Figure 3.16). It combines the Unreal Tournament game engine (responsible for rigid-body physics and content presentation) with an inference engine (responsible for reasoning and updating the scene representation when events occur) and a behavioral engine (responsible for recognizing actions and changing objects in conceptual terms).

ISReal [161] is another tool for creating 3D simulations. The tool leverages semantic concepts, services and hybrid automata to describe behavior of 3D content elements. The tool has a client-server architecture. The client is based on a 3D content presentation tool, e.g., an XML3D browser, while the server is built of several services enabling selection and configuration of content. A graphics environment maintains and renders 3D scene graphs. A global scene environment manages global scene ontologies, which represent the simulations. A verification environment checks spatial and temporal requirements against properties of content objects. An agent environment manages intelligent avatars, e.g., their perception of the scene. The user interface is capable of communicating with web-based and immersive virtual reality platforms.

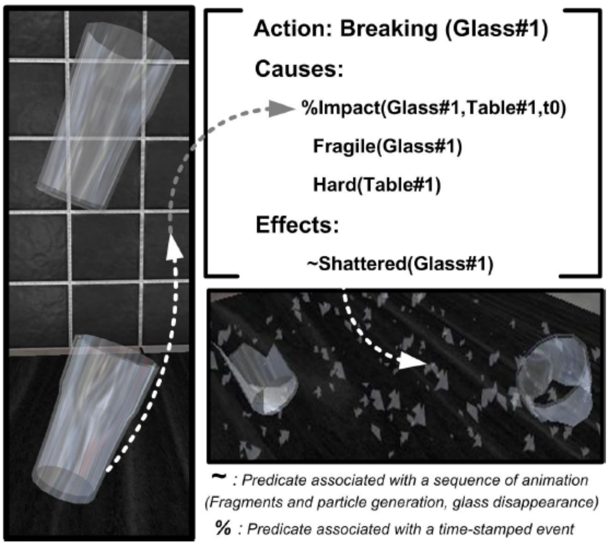

Action: Breaking (Glass#1)
Causes:
%Impact(Glass#1,Table#1,t0)
Fragile(Glass#1)
Hard(Table#1)
Effects:
~Shattered(Glass#1)
~ : Predicate associated with a sequence of animation (Fragments and particle generation, glass disappearance)
% : Predicate associated with a time-stamped event

Figure 3.16: An example of a semantic action representation. Source: [180]. Courtesy of J. L. Lugrin and M. Cavazza

The Simulator X [129, 173, 243] consists of several modules enabling content selection and configuration: content model based on actors, which are the main elements of the created simulation; the state of the simulation stored in shared variables; entities describing actors; binding for the integration of artificial intelligence into the simulation; object-oriented scripts and a multimodal user interface. Another tool for creating 3D simulation with respect to content semantics has been proposed in [194]. The tool enables generation and injection of semantic properties into virtual environments using lexical ontologies such as WordNet [20] or VerbNet [41].

Semantic Modeling of Buildings and Spatial Data

A few tools have been proposed to design 3D models of buildings and spatial data. In [126], a feature-based modeling tool has been described. The tool is an implementation of the modeling paradigm based on universes (cf. Section 3.3). It provides a graphical user interface for modeling content, including selection and configuration of elements, in particular for spatial data modeling and industrial design. The tool leverages an intermediate model that includes both shapes and their semantics. The semantics of features is specified by geometric relations between entities representing features and feature parameters.

In [204], a semantics-based Virtual Design Environment (VDE) has been described. The tool is a graphical application that enables selection and configuration of predefined content elements to create 3D models of buildings. The tool enables the incorporation of different semantic information into the created 3D content. Functional semantics represents the purpose of the design and essential constraints of different parts of the modeled building. Spatial semantics represents the contiguity between different elements of the created model. Esthetical semantics includes visual dominance, symmetry and modality (centrality, linearity and radiality). Environmental semantics covers lightning, acoustic as well as thermal and air ventilation. Contextual semantics includes council regulations, e.g., driveway access.

In [217], a tool for semantic integration of procedural techniques for modeling buildings has been described. The goal of the tool is to generate buildings that are complete (including both external and internal elements such as facade, stairs, furniture, etc.) and congruent (without conflicting elements, e.g., overlapping, with improper properties and functions). The tool includes several modules. Semantic models of buildings are reflected by an ontology (stored in a library) that specifies such conceptual elements as floors, walls, rooms, windows, chairs, etc. A semantic moderator checks consistency of the 3D model being created. Wrapping modules are used to provide a uniform generic access interface to the semantic moderator for different procedural generation modules that are responsible for content selection and configuration. Plans are documents describing the schemes of buildings, while a conductor is used to accomplish the steps of the applicable plan. The tool enables separation of concerns between different modeling users. While the procedural generation modules need to be implemented by a developer, plans of buildings may be further created by an expert in modeling buildings.

Semantic Adaptation of 3D Content

Several works have addressed adaptation of interactive 3D content. Content adaptation encompasses mainly the selection of content elements and properties to be presented as well as transformation of content representations to different content representation formats and languages. In [149, 151], a tool transforming virtual environments for multi-modal interfaces has been proposed. The tool supports such interfaces as PDA, microphone or gestures-based. The tool uses XML-based device descriptors that define parameters of interaction devices (IO channels). Device descriptors are mapped to virtual entity descriptors (e.g., animated shapes, virtual characters and multimedia documents) and a virtual environment descriptor (the world under control). Tool configuration may be completed using an interface based on visual programming, which offers diagrams corresponding to different descriptors and their properties (Figure 3.17).

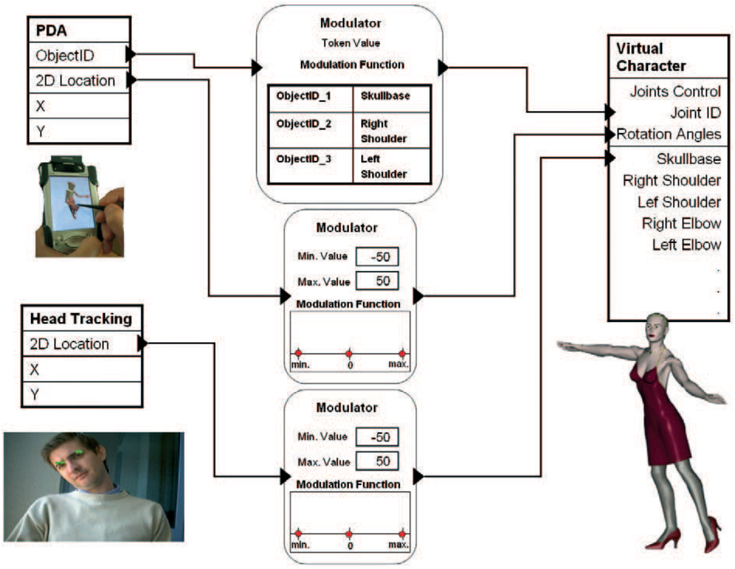


Figure 3.17: Pen and gestures-based interfaces described by a diagram enable semantic manipulation of a 3D object. Source: [151]. Courtesy of M. Gutiérrez, D. Thalmann and F. Vexo

In the Amacont tool [115], the modeling activities: selection, configuration and transformation are available from a graphical interface. Content adaptation is performed according to rules registered in and processed by different modules. Adaptation covers different aspects of the content (appearance, structure, behavior and functionality), which are determined with respect to user preferences, device used and the context of interaction (location, connection parameters, network users, temperature, etc.)—retrieved from sensor elements.

In the 3DAF tool [89], which is based on the 3DSEAM content model, selection and configuration are performed in a flexible way, on demand according to rules registered within the tool (Figure 3.18). Once queries are issued by users invoking methods of the communication interface, the invocations are translated into SQL-like queries and process by a query manager. Next, content objects are retrieved from an annotation repository and a final 3D scene is created.

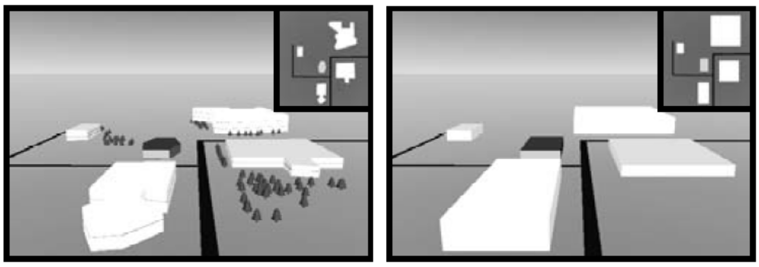


Figure 3.18: Result of the transformation of a campus following semantic adaptation rules. Source: [89]. Courtesy of I. M. Bilasco, M. Villanova-Oliver, J. Gensel and H. Martin

The AttribIt tool [102] has been designed for modeling adaptable 3D content with semantic attributes. In the tool, techniques of information retrieval, machine learning and computer vision are used to extract implicit knowledge from 3D models and to reflect different parts and elements (e.g., related to geometry and appearance) of 3D models by different strengths of attributes, e.g., aerodynamics or scary heads (Figure 3.19). A graphical user interface enables flexible selection of content elements, choice of the desirable strengths of attributes of the elements, and observation of the results. A statistical model permits the simultaneous and coherent change of multiple content elements once the values of attributes change.

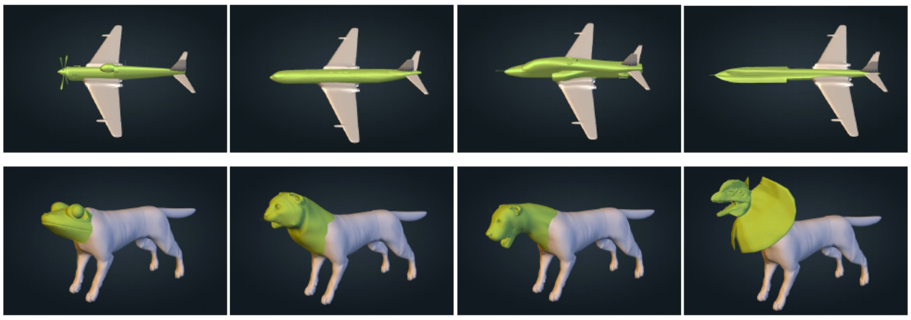


Figure 3.19: Variants of 3D models with different strengths of semantic attributes reflecting aerodynamics and scary heads. Source: [102]. Courtesy of S. Chaudhuri, E. Kalogerakis, S. Giguere and T. Funkhouser

3.7. Summary

Applying semantics to interactive 3D content receives increasing attention in research community as one of the possible approaches to facilitating creation and manipulation of 3D content. Semantic representation and modeling of 3D content is used in various applications and domains, such as cultural heritage, entertainment, CAD and simulation. The available solutions have been classified according to the proposed taxonomy. The following conclusions may be drawn on the basis of the survey.

The majority of the works focus on schemes for semantic representation of 3D content. The schemes are usually XML-based ontologies encoded with the semantic web standards (RDF, RDFS and OWL), which create a common space for representing the semantics of web content. The schemes enable content representation at conceptual and concrete levels of abstraction. Schemes are used within methods and tools for modeling 3D content. The previous works usually address tools for semantic modeling of 3D content. A smaller group of works address methods of modeling, focusing on the way in which semantic content representations are processed to produce final 3D content presentations. The available methods and tools typically do not cover the whole modeling process, but enable only selected modeling activities to be performed in semantic way.

In the available approaches little attention is paid to multi-layered 3D content representation at different levels of abstraction—at the level that is specific to 3D modeling and at levels that are specific to arbitrarily selected applications or domains. Multi-layered representation can enable more flexible content creation and management (indexing, searching and analyzing) than in the case of only concrete or only conceptual representation. Next, the available solutions do not strongly benefit from discovery of tacit knowledge hidden in semantic 3D content representations, which could significantly reduce effort in modeling the content. Furthermore, only few of the available methods and tools permit semantic transformation of content to different 3D content representation languages. In contrast to transformations based on typical grammar analysis, semantic transformation may be more convenient for semantically represented content. It can offer new opportunities related to the meaning of content elements, e.g., transforming specific subgroups of 3D scenes or using common generic transformation rules for different languages. Another gap is related to flexible content adaptation covering queries to semantic content representations and contextual information, e.g., device used, user location and preferences.

4. The SEMIC Approach

In this chapter, the SEMIC approach—*Semantic Modeling of Interactive 3D Content*—is introduced. First, the motivations and requirements for SEMIC are presented, then the general concept of the approach is explained.

4.1. Motivations and Requirements

Although a number of approaches have been proposed for modeling interactive 3D content, they lack general and comprehensive solutions for creating content for modern VR/AR systems. The following shortcomings may be identified in the available approaches.

1. 3D content is typically created using a limited set of predefined concepts (specific to 3D content or specific to an application or domain). Existing approaches do not enable conceptual 3D content creation with arbitrary domain-specific concepts. The approaches lack generic methods of linking domain-specific concepts with 3D content components and properties to enable use of a wide range of ontologies for modeling 3D content.
2. 3D content components and properties are explicitly specified in both imperative and declarative 3D content representation languages. Existing approaches do not provide declarative 3D content representation combined with discovery of tacit knowledge, which would enable inference of new facts based on the stated facts. In particular, complex means of expressing semantics, e.g., restrictions on classes, domains, ranges and cardinality of properties, rules and operations on sets, are not employed in 3D content creation.
3. Existing approaches do not enable separation of concerns in modeling 3D content between different users—IT-specialists as well as experts in particular domains. For example, geometrical shapes and textures may be created by an expert in computer graphics, mapped to domain-specific concepts by a developer and further used by a domain expert to create 3D scenes using only domain-specific objects.
4. 3D content adaptation focuses mostly on contextual selection of values for 3D content properties. Such adaptation typically addresses differences in hardware and software used. It does not cover different aspects of 3D content represented at different levels of abstraction. Moreover, 3D content is adapted according to predefined rules that cannot be combined with 3D content representations into highly-expressive knowledge bases representing new adapted 3D content.
5. Transformation of 3D content representations to different formats is typically performed using individual scripts implementing specific algorithms. The approaches do not enable generic transformation to different formats of 3D content representation. A generic transformation can be performed using an algorithm that is common for different target formats and sets of rules that are specific to particular formats. For instance, VRML and X3D have similar structures but different syntax of documents.

Recent trends in the development of the web provide new opportunities for efficient and flexible content creation, which go beyond the current state of the art in modeling 3D content and enable meeting functional requirements imposed by modern content creation scenarios. The main **requirements** are summarized below.

— **Requirement 1: Declarative modeling** of 3D content, which emphasizes specification of the results to be presented, but not the way in which the results are to be achieved. For instance, create a sphere and a cube such that the cube is closer to the observer than the sphere, instead of—first create the sphere and next create the cube so that a part of the sphere is hidden behind the cube, as seen by the observer.

— **Requirement 2: Conceptual modeling** of 3D content components and properties at arbitrarily chosen levels of abstraction, including aspects that are directly related to computer graphics as well as aspects that are specific to a particular application domain. For instance, a car includes wheels, doors, a windscreen and so on.

— **Requirement 3: Knowledge discovery** leading to revealing 3D content properties, dependencies and constraints, which are not explicitly specified, but which may be extracted from the explicit facts and have impact on the modeled content. For instance, present only sub-classes of guns in a virtual museum.

— **Requirement 4: Separation of concerns** in 3D content creation, which may involve different modeling users with different expertise, who are equipped with different modeling tools, e.g., to facilitate content creation by domain experts who are not IT-specialists. For instance, a graphic artist creates a texture, a technician maps it to a domain-specific concept of material and a furniture designer applies the material to different pieces of virtual furniture.

— **Requirement 5: On-demand customization** of 3D content including selection of particular content objects to be presented, selection of the desirable features and behavior of objects, extension of content objects with new features and behavior as well as composition of selected content elements into more complex content objects. Content customization may be performed regarding user requirements and preferences, the context of interaction, client device used, etc.

— **Requirement 6: Multi-platform representation** of 3D content, which covers different hardware and software platforms. 3D content may be described using different content representation formats and languages, and presented within multiple content browsers and presentation tools.

The fulfillment of the requirements can lead to the development of a new class of efficient methods and systems of creating interactive 3D content that can contribute to wider dissemination and use of 3D content on the web.

4.2. Outline of the SEMIC Approach

The main contribution of this dissertation is the *Semantic Modeling of Interactive 3D Content* (the SEMIC approach) [239]. SEMIC goes beyond the current state of the art in modeling 3D content by satisfying the aforementioned requirements through the use of the semantic web techniques. SEMIC enables declarative creation of 3D content based on different domain-specific ontologies. SEMIC supports knowledge discovery in content representations and separation of concerns between different users in content creation. The created content may be presented on different platforms. The general concept of the approach is presented in Figure 4.1.

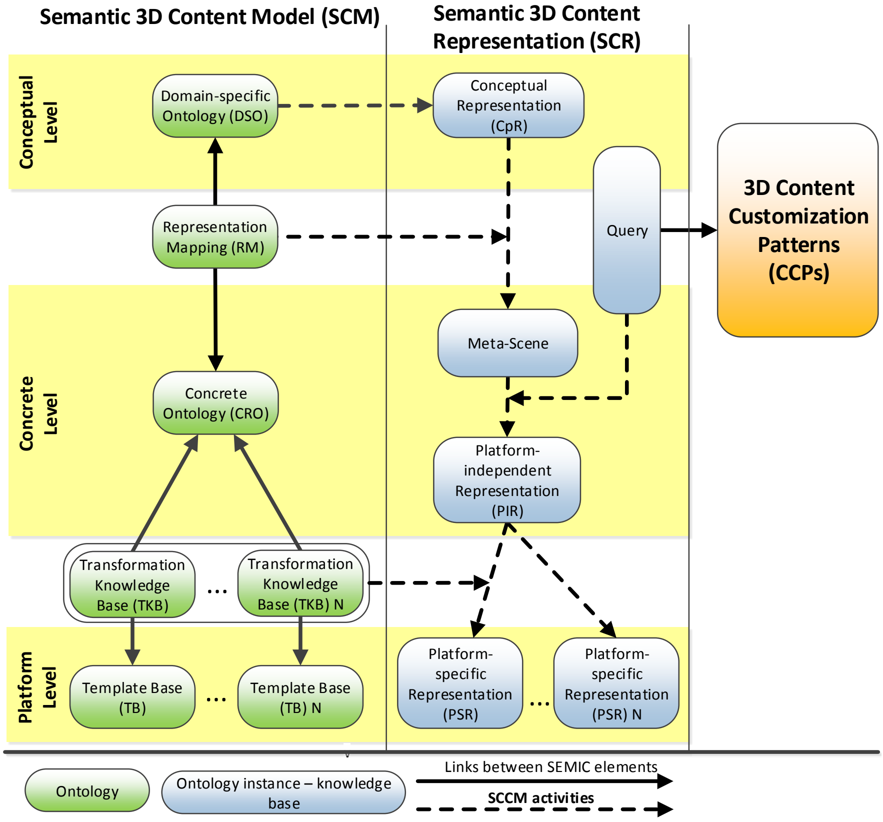

Semantic 3D Content Model (SCM)
Semantic 3D Content Representation (SCR)
Conceptual Level
Concrete Level
Platform Level
Domain-specific Ontology (DSO)
Representation Mapping (RM)
Concrete Ontology (CRO)
Transformation Knowledge Base (TKB)
...
Transformation Knowledge Base (TKB) N
Template Base (TB)
...
Template Base (TB) N
Conceptual Representation (CpR)
Query
Meta-Scene
Platform-independent Representation (PIR)
Platform-specific Representation (PSR)
...
Platform-specific Representation (PSR) N
3D Content Customization Patterns (CCPs)
Ontology
Ontology instance – knowledge base
Links between SEMIC elements
SCCM activities

Figure 4.1: The concept of the *Semantic Modeling of Interactive 3D Content* (SEMIC)

SEMIC consists of two interrelated elements that address different aspects of 3D content creation and representation. The first element—the *Semantic 3D Content Model* (SCM) [131, 134, 138]—provides concepts that enable 3D content representation at different levels of abstraction. In SCM, 3D content is represented at three levels. The *Conceptual Level* is determined by a *Domain-specific Ontology* (DSO), which defines concepts that are abstract in the sense of presentation (are not specific to computer graphics), e.g., buildings, cars, traffic lights, signs, etc. The *Concrete Level* is determined by the *Concrete 3D Content Ontology* (CRO), which defines concepts that are specific to computer graphics, e.g., meshes, light sources, materials, animations, etc. The *Platform Level* is determined by *Template Bases* (TBs) encoded in different (arbitrarily chosen) 3D content representation languages. TBs define concepts that are specific to 3D content representation at different hardware and software platforms, e.g., the `Material` node in VRML, the `X3DMaterialNode` node in X3D and the `Material` class in Java3D.

The links between the neighboring levels are specified within ontologies and knowledge bases. Different DSOs are mapped to the CRO by *Representation Mappings* (RMs), which are knowledge bases. For instance, a car may be mapped to a set of meshes covered with textures, changes in the time of day may be mapped to an animation of the intensity of a light source, etc. The CRO is mapped to different TBs by different *Transformation Knowledge Bases* (TKBs). For instance, the `cro:Navigation` class of the CRO is mapped to the `NavigationInfo` element of VRML and to a fragment of code of ActionScript in 3D Adobe Flash presentations. SCM permits the use of domain- and application-specific knowledge to simplify the content creation process at a high level of abstraction by domain experts who are not IT-specialists. Knowledge, which is expressed at different levels of abstraction/semantics, is modeled using linked (solid lines) ontologies and knowledge bases.

SCM is used within the second element of SEMIC—the *Semantic 3D Content Creation Method* (SCCM) [137, 139]. SCCM is a sequence of steps (dashed lines in Figure 4.1), in which the concepts of the SCM model are used to create 3D content. Some of the steps of the SCCM method are accomplished manually by a human, whereas other steps are accomplished automatically—by software that transforms 3D content representations, enabling transitions between the subsequent steps.

In the SCCM method, the domain specific ontologies (DSOs) are used for designing *Conceptual 3D Content Representations* (CpRs), which reflect content at the conceptual level in the forms specific to arbitrarily selected domains. CpRs are processed with respect to RMs and transformed to *meta-scenes*. A meta-scene is a generalized 3D content representation, which reflects content at both conceptual (high) and concrete (low) levels of abstraction. Meta-scenes may be customized in terms of the content components and properties that should be provided to a content consumer (user or application). Requirements for the customization of a meta-scene are specified within queries that are prepared by content consumers. For instance, a meta-scene of a city includes models of buildings and animated models of cars. It may be customized into a scene that includes only buildings, buildings and cars without animations, etc. Customization queries are created according to the *3D Content Customization Patterns* (CCPs) [238]. A *Platform-independent 3D Content Representation* (PIR) generated on the basis of a meta-scene and a query is transformed to a *Platform-specific 3D Content Representation* (PSR), which may be presented at particular hardware and software platforms, e.g., smartphones equipped with browsers supporting WebGL, workstations equipped with X3D browsers, etc. PSRs are generated using TKBs.

SEMIC is firmly based on the semantic web standards (RDF, RDFS and OWL). The restrictive use of the formally specified semantic web standards is preferred over the use of other concepts (in particular rules, which are flexible to use and have high semantic expressiveness) because of the following two reasons. First, the semantic web standards provide concepts that are widely accepted on the web and can be processed using well-established tools, such as editors and reasoners. Second, complexity measures have been investigated and specified for these standards, including a number of typical reasoning problems (such as ontology consistency, instance checking and query answering) [224], which permits building applications with more predictable computational time. The particular elements of SEMIC are presented in detail in the following sections.

5. The SEMIC Semantic 3D Content Model

In this chapter, the SEMIC *Semantic 3D Content Model* (SCM) is described [131, 134, 138–141]. SCM enables declarative, platform-independent representation of 3D content at different levels of abstraction. In this chapter, first, an overview of SCM and formal definitions that are common to different elements of SCM are presented. Second, particular elements of SCM are described in detail.

5.1. Overview of the SCM Model

SCM is a set of ontologies that enable creation of *Semantic 3D Content Representations* (SCRs) at different levels of abstraction (Figure 5.1). An SCR is a knowledge base that is compliant with SCM. SCM encompasses individual ontologies that are specific to different aspects of 3D content. The ontologies enable creation of partial semantic content representations (knowledge bases) included in an SCR. The semantic content representations reflect the content in a declarative way—by specifying content elements, their properties and relations between them (Chapter 4/Requirement 1).

SCM consists of four ontologies as described below.

1. The *Concrete 3D Content Ontology* (CRO) enables direct representation of 3D content taking into account different content modalities (e.g., visual, aural or haptic). In this dissertation, visual presentations of content, which are based on computer graphics are addressed. Such presentations include, among others geometrical components, materials, textures and animations. The CRO is used to build *Concrete 3D Content Representations* (CrRs).
2. A *Domain-specific Ontology* (DSO) enables high-level 3D content representation based on domain-specific classes and properties. The classes and properties are abstract in the sense of their final presentation and, in general, are not specific to 3D computer graphics, e.g., buildings, cars, artifacts and animals. A DSO is used to build *Conceptual 3D Content Representations* (CpRs). A CpR is a knowledge base that represents 3D content at a conceptual level of abstraction. SCM may include different DSOs. SCM introduces neither requirements for DSOs nor restrictions on DSOs that can be used.
3. The *Mapping Ontology* (MO) enables mapping of domain-specific classes and properties (included in a DSO) to concrete classes and properties (included in the CRO). Mapping is expressed by *Representation Mappings* (RMs), which are knowledge bases that conform to the MO. RMs permit modeling 3D content with domain-specific classes and properties that are not related to computer graphics.
4. The *Transformation Ontology* (TO) enables transformation of statements on concrete components and concrete properties to code fragments encoded in a particular 3D content representation language (Java, ActionScript, X3D, etc.). A particular transformation is described by a *Transformation Knowledge Base* (TKB), which conforms to the TO, and a *Template Base* (TB), which includes parametrized code templates encoded in a 3D content representation language.

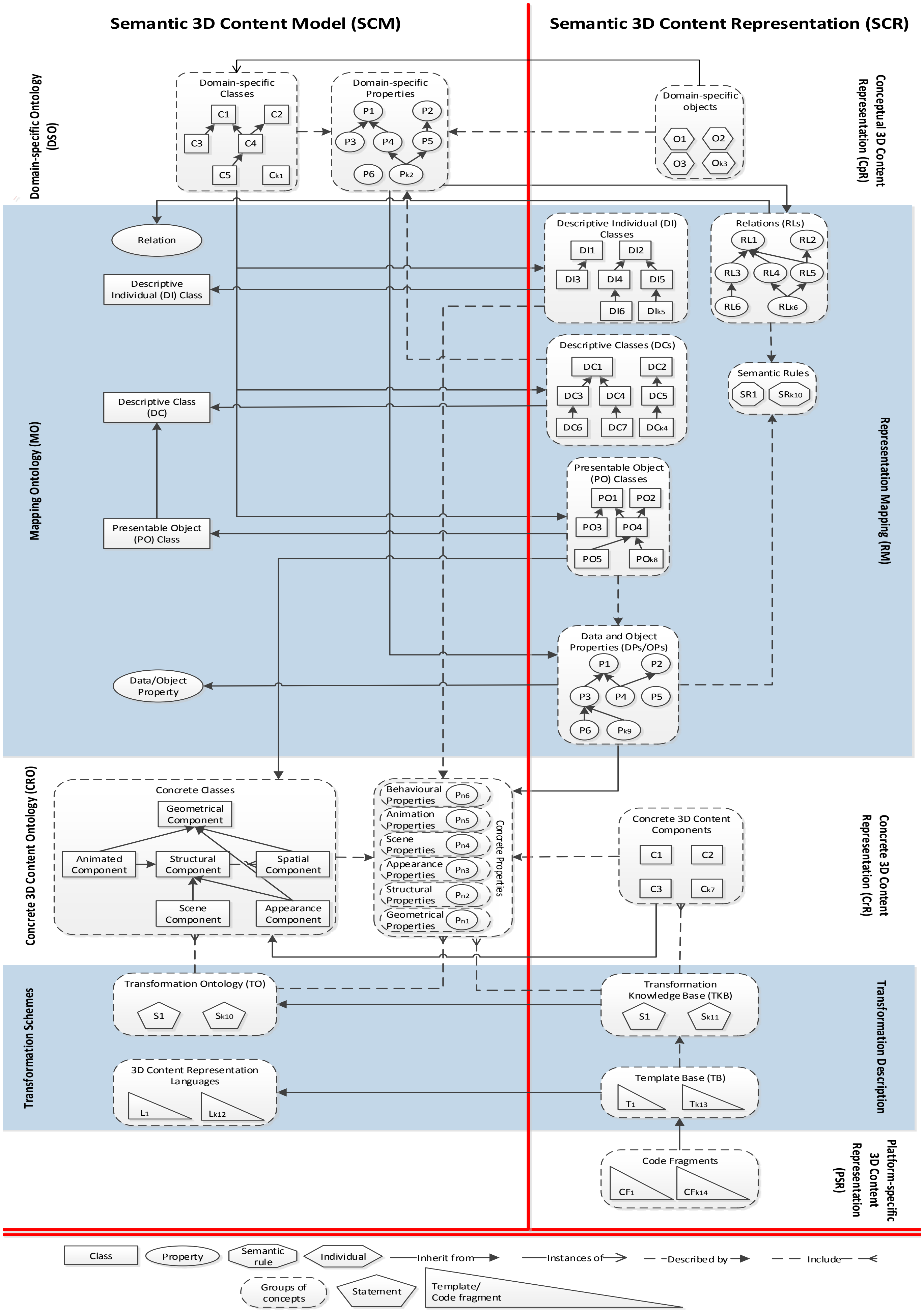


Figure 5.1: The *Semantic 3D Content Model* (SCM)

The concepts of *3D content*, *3D object* and *3D scene*, which are used across this dissertation are defined as follows.

Definition 1. ***3D content** is a 6-tuple* $\{\mathcal{G}, \mathcal{ST}, \mathcal{P}, \mathcal{SC}, \mathcal{A}, \mathcal{B}\}$*, where:*

- $\mathcal{G}$ *is a set of 2D and 3D geometrical elements, such as meshes, spheres, rectangles and arcs;*
- $\mathcal{ST}$ *is a set of structural elements (e.g., structures of multiple geometrical elements) as well as transformations (e.g., rotation, scale, screw and intersection);*
- $\mathcal{P}$ *is a set of presentational elements and properties, such as textures, colors and light sources;*
- $\mathcal{SC}$ *is a set of elements related to spatial exploration of the content, such as viewpoints and navigation modes;*
- $\mathcal{A}$ *is a set of animations, such as changes of colors, coordinates, normals and positions;*
- $\mathcal{B}$ *is a set of behavioral elements describing interactions between 3D content elements as well as interactions between a user and 3D content elements, such as sensors and events.*

An example of *3D content* is a set of transformed meshes with textures, animations and interactions. Specific kinds of 3D content are *3D object* and *3D scene*.

Definition 2. *A **3D object** is such 3D content that its* $\mathcal{G}$ *contains:*

- *a single 2D or 3D geometrical element, or*
- *multiple 2D and 3D geometrical elements that have a common root element.*

Examples of *3D objects* consisting of only a single geometrical element are a mesh, a cone and a sphere. An example of a *3D object* consisting of multiple elements with a common root is a complex structural element representing an engine with pistons and valves.

Definition 3. *A **3D scene** is such a 3D object that its* $\mathcal{SC}$ *contains viewpoints and navigation modes.*

Due to the presence of viewpoints and navigation modes, a *3D scene* may be viewed by an observer. *3D scenes* typically have complex structures consisting of a number of geometrical elements. Examples of *3D scenes* are: a virtual museum with exhibitions, a room with avatars in a game, a court with players, etc.

A **3D content representation** is a dataset describing the *3D content* in a way that is sufficient to present the *3D content*. A *3D content representation* may be encoded using:

- 3D content representation formats, e.g., obj [54], 3ds [17], VRML [222] and X3D [223];
- 3D content representation languages, e.g., C++, Java, JavaScript and ActionScript with appropriate libraries, e.g., OpenGL [9], [186], WebGL [69] and Away3 [2];
- database models, e.g., X-VRML [234] and X-VRDB [233];
- semantic web techniques, e.g., RDF [225], RDFS [226] and OWL [224].

Examples of *3D content representations* are: a VRML document, an X3DOM script embedded in a web page, compiled classes based on Java3D [186] responsible for drawing shapes, and an X-VRML template for dynamic 3D content generation. In the remainder of this dissertation, the focus is on *semantic 3D content representations* that are encoded using the semantic web standards.

5.2. Semantic Web Definitions

SCM is based on a number of concepts defined in the specifications of the RDF [225], RDFS [62] and OWL [56, 57] semantic web standards:

- **rdfs:Class** [62], which is the set of all RDF classes (referred to as **classes** in this dissertation);
- **rdfs:Resource**, which is the *class* of all RDF resources (referred to as **resources** in this dissertation) [62];
- **rdf:Property**, which is the *class* of all RDF properties (referred to as **properties** in this dissertation) [225];
- **rdf:type** [225], which is referred to as **type** in this dissertation;
- **rdfs:Literal**, which is the *class* of all RDF literal values (referred to as **literal values** in this dissertation) [62], e.g., strings, integers and floats;
- **rdfs:subClassOf** [62], which is referred to as **subclass** in this dissertation. A *class* that has a *subclass* is referred to as a **superclass** of the *subclass* in this dissertation;
- **rdfs:subPropertyOf** [62], which is referred to as **subproperty** in this dissertation. A *property* that has a *subproperty* is referred to as a **superproperty** of the *subproperty* in this dissertation;
- **rdfs:domain** and **rdfs:range** of a *property* [62], which are referred to as **domain** and **range** in this dissertation;
- **owl:DatatypeProperty**, which is a subclass of rdf:Property [57]. Its instances are referred to as **data properties** in this dissertation;
- **owl:ObjectProperty**, which is a subclass of rdf:Property [57]. Its instances are referred to as **object properties** in this dissertation;
- **owl:NamedIndividual** [56], which is a *class* of all OWL named individuals (referred to as **individuals** in this dissertation).

To express knowledge, *resources* are used within **statements**. A *statement* expresses a fact. It consists of a **subject**, a **property** and an **object**. The *subject* is a *resource* being described. The *object* is a *resource* that describes the *subject*. The *property* specifies a relation between the *subject* and the *object*. An example of a *statement* is: a sensor (subject) is assigned to (property) a mesh (object).

Following [225], a **statement** on subject S, property P and object O is a triple <S,P,O>, where:

- S is a *resource*,
- P is a *property*,
- O is a *resource*.

A statement **is true** (**satisfied**) if and only if the *subject* and the *object* are related by the *property*. The existence of a *property* P between *resources* X and Y is denoted as: $P(X,Y)$—X and Y are **related** by P. In SEMIC, the *open world assumption* [208] is used, if not stated otherwise. Hence, a *statement* that is specified is true. A *statement* that is not specified is not considered false.

Let $\mathcal{S}$ denote the **set of all possible statements**.

Multiple *statements* may be combined into a **rule**. A *rule* is interpreted as an implication, which makes the satisfaction of some *statements* (contained in the *head*) conditional on the satisfaction of other *statements* (contained in the *body*).

Following [227], a **rule** is a pair <*body*, *head*>, where:

- the *body* and the *head* are sets of statements, and
- if the conjunction of the statements of the *body* is true, then the conjunction of the statements of the *head* is also true.

$R =$ <body, head> :
$body = \{Sb_1, ..., Sb_n : \forall i \in \{1, ..., n\} : Sb_i \in \mathcal{S}\} \wedge$
$head = \{Sh_1, ..., Sh_m : \forall j \in \{1, ..., m\} : Sh_j \in \mathcal{S}\} \wedge$
$Sb_1 \wedge ... \wedge Sb_n \Rightarrow Sh_1 \wedge ... \wedge Sh_m$
If *body* is empty, the *rule* is a statement.

Let $\mathcal{R}$ denote the **set of all possible rules**.

An example of a *rule* is: if the x dimension of a box is greater then the y dimension (*body*—statement 1) and the y dimension is greater then the z dimension (*body*—statement 2), the box is put on the side with the edges x and y (*head*—statement 1) and its color is red (*head*—statement 2).

5.3. SCM Definitions

Multiple *rules* form an *SCM knowledge base*.

Definition 4. *An **SCM knowledge base** is a non-empty set of rules.*

Rules in *SCM knowledge bases*, describe *classes*, *properties* and *individuals*.

Let $\mathcal{KB}$ denote the **set of all possible SCM knowledge bases**.

The **union of SCM knowledge bases** is also an *SCM knowledge base*.
$\forall KB_1 \in \mathcal{KB} \quad \forall KB_2 \in \mathcal{KB}(KB_1 \cup KB_2 \in \mathcal{KB})$

A specific kind of *SCM knowledge base* is *SCM ontology* in which only *classes* and *properties* are described. Unlike in *SCM knowledge bases*, in *SCM ontologies* no *individuals* are described.

Definition 5. *An **SCM ontology** is such an SCM knowledge base that is a set of rules whose bodies and heads have subjects that are either classes or properties.*
$O \in \mathcal{KB} : \quad \forall R \in O(R = \text{<body, head>} \wedge \forall S \in body \cup head$
$(S = \text{<sub,prop,obj>} \wedge (rdf{:}type(sub, rdfs{:}Class) \vee rdf{:}type(sub, rdf{:}Property))))$

Definition 6. *Resource R is **defined in** an SCM knowledge base KB as an **instance of** a class C if and only if there is a statement S in the KB such that the subject of S is R, the property of S is rdf:type and the object of S is C.*
$\forall R \in rdfs{:}Resource \quad \forall KB \in \mathcal{KB} \quad \forall C \in rdfs{:}Class$
$(definedIn(R, KB) \wedge instanceOf(R, C) \quad \Leftrightarrow \quad \exists S \in KB(S = \text{<R,rdf:type,C>})$

An *SCM knowledge base* KB_1 is an *instance of* an *SCM knowledge base* KB_2 if and only if KB_1 uses a *class* or a *property defined in* KB_2.

Definition 7. *An SCM knowledge base KB_1 is an **instance of** an SCM knowledge base KB_2 if and only if there is such a statement in KB_1 that either the subject or the property or the object of the statement is defined in KB_2.*
$\forall KB_1 \in \mathcal{KB} \quad \forall KB_2 \in \mathcal{KB}(instanceOf(KB_1, KB_2) \quad \Leftrightarrow \quad \exists S \in KB_1$
$(S = \text{<sub,prop,obj>} \wedge (definedIn(sub, KB_2) \vee definedIn(prop, KB_2) \vee definedIn(obj, KB_2))))$

SCM consists of several *SCM ontologies*.

Definition 8. *The **Concrete 3D Content Ontology (CRO)** is such an SCM ontology that:*
$CRO = GL \cup STL \cup APL \cup SCL \cup ANL \cup BL$, *where:*

— *GL (**geometry layer**) is an SCM ontology in which classes and properties related to the geometry of 3D content are defined,*

— *STL (**structure layer**) is an SCM ontology in which classes and properties related to the structure of 3D content are defined,*

— *APL (**appearance layer**) is an SCM ontology in which classes and properties related to the appearance of 3D content are defined,*

— *SCL (**scene layer**) is an SCM ontology in which classes and properties related to spatial exploration of 3D content are defined,*

— *ANL (**animation layer**) is an SCM ontology in which classes and properties related to animations of 3D content are defined,*

— *BL (**behavior layer**) is an SCM ontology in which classes and properties related to the behavior of 3D content are defined.*

The *layers* correspond to distinct aspects of 3D content. The layers are described in Section 5.4.2.

Definition 9. *The **Mapping Ontology (MO)** is a 5-tuple* $\{RPOC, RDC, RMP, RDIC, RRLC\}$*, where:*

— *RPOC is the Root Presentable Object Class from which all Presentable Object Classes (described in Section 5.6.3) inherit;*

— *RDC is the Root Descriptive Class from which all Descriptive Classes (described in Section 5.6.3) inherit;*

— *RMP is the Root Mapping Property from which all Mapping Properties (described in Section 5.6.3) inherit;*

— *RDIC is the Root Descriptive Individual Class from which all Descriptive Individual Classes (described in Section 5.6.3) inherit;*

— *RRLC is the Root Relation Class from which all Relations (described in Section 5.6.3) inherit.*

Mapping a DSO to the CRO is described in Section 5.6.

Definition 10. *The **Transformation Ontology (TO)** is a 5-tuple* $\{SCPC, SPC, TSPC, TPC, TPPC\}$*, where:*

— *SCPC is the Statement Collection Pattern Class of all Statement Collection Patterns (described in Section 5.7.2);*

— *SPC is the Statement Pattern Class of all Statement Patterns (described in Section 5.7.2);*

— *TSPC is the Template Set Pattern Class of all Template Set Patterns (described in Section 5.7.2);*

— *TPC is the Template Pattern Class of all Template Patterns (described in Section 5.7.2);*

— *TPPC is the Template Parameter Pattern Class of all Template Parameter Patterns (described in Section 5.7.2).*

Multi-platform 3D content representation based on TO is described in Section 5.7.

Definition 11. *The **Core Semantic 3D Content Model (CSCM)** is such an SCM ontology that:*
$CSCM = CRO \cup MO \cup TO$*, where:*

— *CRO is the Concrete 3D Content Ontology,*

— *MO is the Mapping Ontology,*

— *TO is the Transformation Ontology.*

A **Domain-specific Ontology (DSO)** is an SCM ontology that describes 3D content in a way that is specific to an application or domain and enables presentation of the content. Various *domain-specific* *ontologies* can be used within SCM to represent 3D content. The role of *domain-specific ontologies* is described in Section 5.5.

Definition 12. *The **Semantic 3D Content Model (SCM)** is such an SCM ontology that:*
$SCM = CSCM \cup DSO$*, where:*

— *CSCM is the Core Semantic 3D Content Model,*
— *DSO is a Domain-specific Ontology.*

Definition 13. ***Semantic 3D Content Representation (SCR)** is such an SCM knowledge base that is an instance of SCM.*

Definition 14. *An **SCM class** is a class defined in SCM or in a SCR.*

Like *classes*, *SCM classes* may form hierarchies. For example, every *cone* (*subclass*) is a *shape* (*superclass*).

Definition 15. *An **SCM individual** is an instance of an SCM class.*

Different *SCM classes* of *SCM individuals* are distinguished (e.g., transformations, meshes and animations).

Definition 16. *An **SCM property** is a property defined in SCM or in a SCR.*

SCM properties are used to describe *SCM classes* and *SCM individuals*. Like *properties*, *SCM properties* may form hierarchies. For example, a car that *runs on* (*subproperty*) a road *is on* (*superproperty*) the road. Two kinds of *SCM properties* are distinguished depending on the *range—SCM data properties* and *SCM object properties*.

Definition 17. *An **SCM data property** is an SCM property that is a data property.*

Definition 18. *An **SCM object property** is an SCM property that is an object property.*

Examples of the use of *SCM data* and *object properties* are: a sphere (an *SCM individual*) has radius (an *SCM data property*) 10 (a *resource* that is a *literal value*); a mesh (an *SCM individual*) has (an *SCM object property*) a material (a *resource* that is an *SCM individual*).

Definition 19. *An **SCM resource** is an SCM class or an SCM individual or an SCM property.*

SCM resources are used in SEMIC to represent 3D content.

5.4. Concrete 3D Content Representation

In this section, the *Concrete 3D Content Ontology* (CRO) and *Concrete 3D Content Representation* (CrR) are explained. First, an overview of the CRO and CrR is provided. Second, the formal model of CrRs is described. Third, elements of the CRO and a CrR are presented in detail.

5.4.1. Overview of Concrete 3D Content Representation

A CrR is an SCM knowledge base that is an instance of the CRO [138]. While the CRO specifies SCM properties and SCM classes that do not have concrete properties assigned, a CrR extends the CRO by specifying concrete components, which are subclasses of SCM classes defined in the CRO, that have SCM properties assigned. Concrete components have particular values of concrete properties assigned. Hence, they are sufficient for building 3D content representations.

The CRO introduces concepts (SCM classes and SCM properties) for low-level 3D content representation (Chapter 4/Requirement 2). Concepts related to distinct aspects of 3D content are separated into several layers (Figure 5.2). The SCM classes and SCM properties defined in the CRO are widely used in well-established 3D content representation languages and programming libraries, such as X3D, VRML, Java3D and Away3D. Examples of CRO classes are: `cro:GeometricalComponent`, `cro:Material` and `cro:NormalInterpolator` (the `cro` prefix indicates the definition of the class in the CRO). Examples of CRO properties are: `cro:coordinates`, `cro:specularColor` and `cro:interpolationPeriod`. In the current form, the CRO contains concepts that are sufficient for designing majority of 3D content, but it could be extended with new concepts depending on particular specific requirements. Although in general, the CRO could address different modalities of content (including visual, aural or haptic elements), in this dissertation the focus in on the geometric modality of content presentation.

5.4.2. Concrete Modeling Layers

The CRO consists of six layers, in which different SCM classes and SCM properties of 3D content are defined—*geometry layer*, *structure layer*, *appearance layer*, *scene layer*, *animation layer* and *behavior layer*. The subsequent layers are partly dependent—every layer uses only its concepts and concepts defined in the lower layers (gray arrows), i.e., 3D content may encompass concrete components of a particular layer without referring to its higher layers. For instance, a complex 3D scene with behavior covers all the layers of the CRO; reusable structural 3D objects without appearance that are to be injected into different complex 3D scenes can be created at layer 2. The layered structure of 3D content imposed by the CRO reduces the complexity and the number of relations between concrete components incorporated in different layers of a CrR. It facilitates the exchange of particular concrete components and the overall creation of 3D content. In addition, possible profiles of CRO implementation may cover only layers reflecting the required aspects of 3D context and their lower layers, e.g., only geometry, geometry and structure, etc.

During the creation of a CrR, CRO classes (*components*) are extended by inheritance to *concrete components*. To simplify the presentation of the layers, components are referred to using their URIs (e.g., `cro:DirectionalLight`), whereas the instances of components are referred to using lowercase letters (e.g., a `spatial component` is an instance of `cro:SpatialComponent`, `textures` are instances of `cro:Texture`).

The Geometry Layer

The geometry layer is the base layer of SCM. It consists of components and SCM properties of 2D and 3D geometrical shapes that form 3D content. The primary `cro:GeometricalComponent` is abstract, so that its descendants are used to represent 3D content. Different components related to geometry are defined in the layer, e.g., arcs, circles, cylinders, boxes and meshes. Since the components of this layer have no common point of spatial reference, their spatial SCM properties (position and orientation) are not given.

As this layer does not address other aspects of 3D content (structure, appearance, scene, animation and behavior), it only enables modeling of separated shapes (3D models of sculptures, buildings, furniture, etc.). Hence, the layer is insufficient for building 3D content for practical presentations.

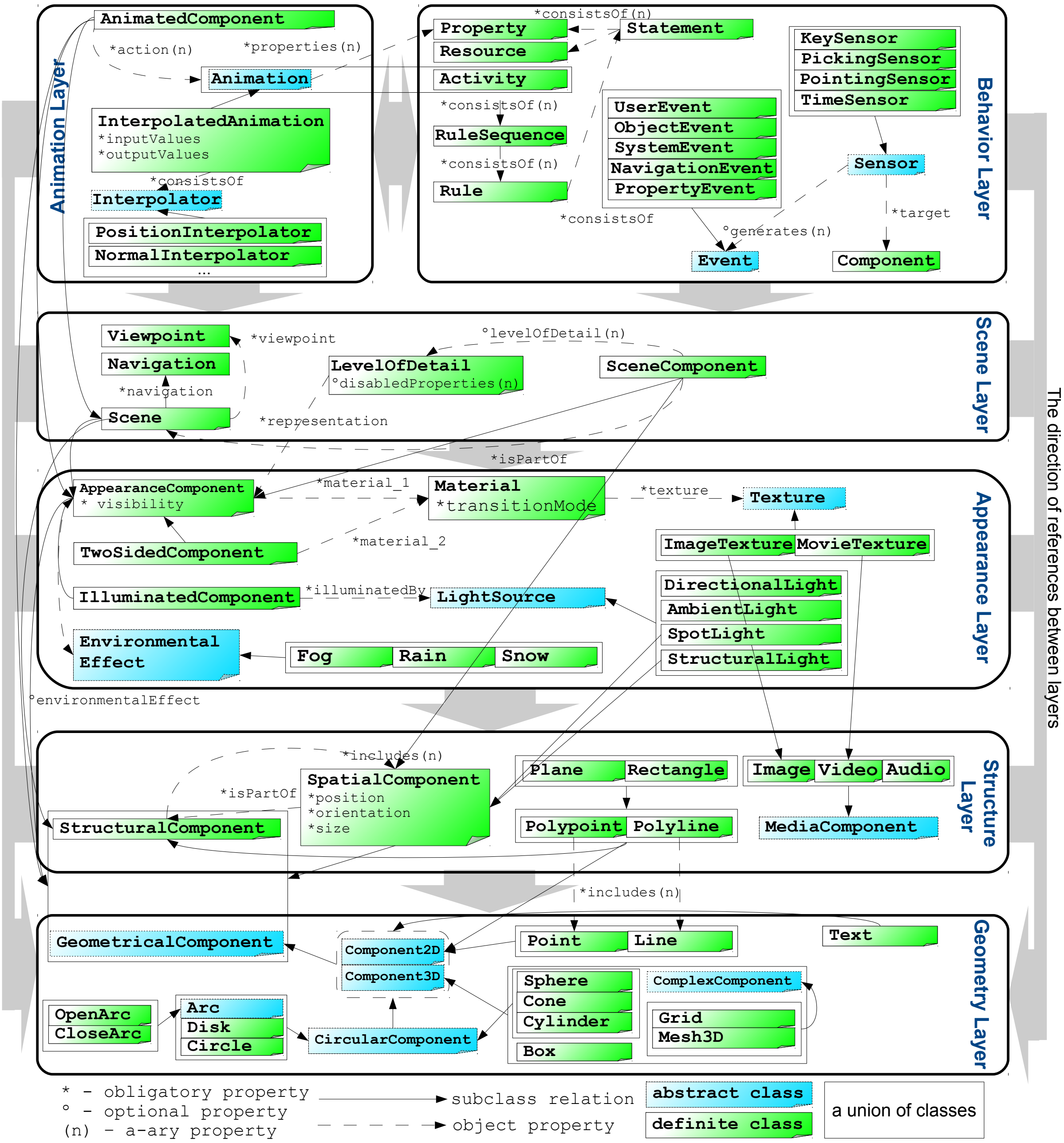


Figure 5.2: The *Concrete 3D Content Ontology* (CRO)

The Structure Layer

The structure layer is the second layer of the CRO. It depends only on the geometry layer. While geometrical components are used to represent basic shapes, structural components that are spatial combinations of multiple geometrical components, are used to represent complex 3D content. Moreover, structural components may recursively include other structural components as well as media resources (images, audio and video) and spatial components. A spatial component is a geometrical component or a structural component, and it has spatial SCM properties (`cro:position`, `cro:orientation` and `size`), which are set relatively to its parent structural component.

Due to the specific meaning of the `cro:include` inverse functional SCM property, structural components may be considered as a whole while assigning some SCM properties (subproperties of the cro:structurallyTransitive property) in higher layers. In particular, a common appearance may be set for all the subcomponents of a structural component, e.g., a car that comprises a number of independent 3D meshes, is as a whole covered with a yellow material with specular highlights.

The components of this layer have no appearance and thereby they are insufficient for creating 3D content in practical cases. However, they describe the structure of 3D content and are used in higher layers, e.g., when determining animations and behavior.

The Appearance Layer

The appearance layer aims at adding appearance to instances of cro:GeometricalComponent and cro:StructuralComponent, which are defined in the lower layers. An appearance component may be either a single- or a two-sided object. Each side can be covered with textures (images or movies) or described by appearance SCM properties (e.g., cro:color and cro:transparency). Common appearance SCM properties may be set for whole structural components by specifying cro:transitionMode—to ignore or respect individual SCM properties of the subcomponents. In addition, appearance components may be illuminated by light sources (e.g., directional, ambient, spot and structural) and enriched with environmental effects.

At this layer, 3D content is represented using components with appearance that have no animations and behavior. Such 3D content may be sufficient for static presentations of individual models, e.g., museum artifacts, furniture or buildings.

The Scene Layer

The primary component of the scene layer is the cro:Scene, which is a subclass of the union of cro:AppearanceComponent and cro:StructuralComponent. A 3D scene has viewpoints and navigation modes assigned. A background and environmental effects of a scene may be specified at the lower appearance layer. A scene may be presented with different levels of detail depending on the current distance to the observer. Each level of detail indicates a set of appearance components included in the scene that should be visible and a set of appearance SCM properties to be disabled for the components.

At this layer, designing complex navigable 3D scenes (without animations and behavior) is feasible. Hence, the layer is sufficient for creating 3D content with basic interaction with a user (navigation), e.g., virtual museum rooms and 3D models of cities.

The Animation Layer

The animation layer consists of components and SCM properties that introduce animations to 3D content representations embedded in the lower layers. As animation may be related to different aspects of 3D content, the layer refers to all the lower layers. cro:AnimatedComponent extends (by inheritance) the union of cro:GeometricalComponent, cro:StructuralComponent, cro:AppearanceComponent and cro:Scene with an arbitrary number of animations. Animations have input and output SCM properties given. An interpolated animation specifies sets of values of input and output SCM properties. The changes of discrete attributes (e.g., texture path and navigation mode) are stepwise. For numeric attributes with continuous domains (e.g., position, scale and color), intermediate input and output values that are not explicitly specified, are calculated from their neighboring values.

3D content represented at this layer covers animated 3D scenes and 3D objects, e.g., moving artifacts, light sources changing colors, morphing, etc.

The Behavior Layer

The behavior layer consists of components and SCM properties that are used to describe behavior and logic of instances of components that are specified in the other layers. The layer has been designed according to the rule-based approach, which enables complex declarative description and reasoning on 3D content. The primary component of behavior descriptions is `cro:Rule`. A rule is interpreted as an implication (if the body is satisfied then the head is also satisfied). Rules without a body (statements) are convenient for describing the state of instances of components. Rules with bodies are mainly used to describe sequences of rules. In a rule sequence, the head of a rule is the body of the next rule. A rule sequence permits an ordered performance of consecutive steps, like in imperative programming. In turn, several independent sequences may create an activity. Activities that depend on events, enable programming interactions (user, object, navigation or system interactions). Events are generated by sensors: key sensors or pointing sensors—for user interactions, collision sensors—for object interactions, time sensors—for system interactions. `cro:Event` and `Sensor`, as well as the SCM properties defined on them in the CRO are similar to the corresponding concepts that are widely-used in other languages for describing 3D content, thus they are not presented in detail in Figure 5.2.

At this layer, the created 3D content covers dynamic 3D scenes and 3D objects with all the aspects addressed in the CRO. The 3D content may change in time due to user interactions, system interactions as well as interactions between components, etc.

5.4.3. Formal Model of Concrete 3D Content Representation

The CRO is the basis for building CrRs. The common *superclass* of all the *SCM classes* that are defined in the CRO is `cro:Component`.

Definition 20. *A **3D content component** is such a subclass of* `cro:Component` *that is defined in the CRO.*

Two kinds of CRO *classes* are distinguished—*abstract classes* and *definite classes*. In SEMIC, *3D content* is comprised of *instances of definite classes*, as *abstract classes* lack properties required for 3D content presentation. For example, `cro:CircularObject` is an *abstract* class, since different circular components are possible, e.g., disks, circles, spheres, etc. Its *subclass* `cro:Circle` is a *definite class*, as it specifies all required properties to draw a *component* instance.

Definition 21. *A **concrete property** of 3D content is such an SCM property that is defined in the CRO and its domain is a subclass of* `cro:Component`*.*

Definition 22. *A **concrete data property** of 3D content is such a concrete property that is an SCM data property.*

Definition 23. *A **concrete object property** of 3D content is such a concrete property that is an SCM object property.*

Definition 24. *A **Concrete 3D Content Representation (CrR)** is such an SCM ontology that is an instance of the CRO.*

In a CrR, *concrete 3D content components* are defined.

Definition 25. *A **concrete 3D content component** is such an SCM class that is a subclass of definite classes and a subclass of restrictions on concrete properties.*

Two kinds of logical *constraints* are distinguished.

Following [224], a **value constraint** on *SCM property* P and *resource* R is a pair <P,R>.

Following [224], a **cardinality constraint** on *SCM property* P and *resource* R is a triple <P,R,N>, where N is an integer.

Following [224], a **restriction** with *constraint* C is an *SCM class* of *resources* for which C is satisfied.

$\mathcal{C}$ is the **class of all possible constraints**. Different *subclasses* of constraints are distinguished.

Following [224], a **has-value constraint** C=<P,R> is a *value constraint* that **is satisfied** for a *resource* R_2 if and only if R is related to R_2 by P.
$\forall C \in \mathcal{C}_{has-value} \quad \forall R_2 \in \text{rdfs:Resource}(satisfiedFor(C,R_2) \Leftrightarrow C = \text{<P,R>} \wedge P(R_2,R))$

Following [224], a **has-value restriction** is a *restriction* with a **has-value constraint**. *Has-value restrictions* are typically used to specify features of *concrete components*. For example, `crr:StatueComponent` (defined in a CrR) is a *concrete component* that is a *subclass* of (extends) `cro:Mesh3D` and a *subclass* of a *has-value restriction* with a *has-value constraint* on the `cro:coordinates` *concrete property*, which determines the exact shape of statues.

Following [224], a **some-values-from constraint** C=<P,R> is a value constraint that **is satisfied** for a *resource* R_2 if and only if at least one *resource* X related to R_2 by P is an instance of R.
$\forall C \in \mathcal{C}_{some-values} \quad \forall R_2 \in \text{rdfs:Resource}(satisfiedFor(C,R_2) \Leftrightarrow C = \text{<P,R>} \wedge \exists X(P(R_2,X) \wedge X \in R))$

Following [224], a **some-values-from restriction** is a *restriction* with a **some-values-from constraint**. *Some-values-from restrictions* are typically used in the SCCM method in content customization to distinguish SCM individuals that are related to other SCM individuals by some SCM properties. For example, SCM individuals that have an animation assigned are animated objects.

Following [224], an **all-values-from constraint** C=<P,R> is a value constraint that **is satisfied** for a *resource* R_2 if and only if any *resource* X related to R_2 by P is an instance of R.
$\forall C \in \mathcal{C}_{all-values} \quad \forall R_2 \in \text{rdfs:Resource}(satisfiedFor(C,R_2) \Leftrightarrow C = \text{<P,R>} \wedge \forall X(P(R_2,X) \Rightarrow X \in R))$

Following [224], an **all-values-from restriction** is a *restriction* with an **all-values-from constraint**. *All-values-from constraints* are typically used to indicate specific concrete *subcomponents* that are to be related to the *concrete components*. For example, all the elements included in an instance of `cro:StructuralComponent` are instances of `cro:SpatialComponent`, as they have spatial SCM properties (position and orientation) that are relative to their parents.

Following [224], a **minimum-cardinality constraint** C=<P,R,N> is a cardinality constraint that **is satisfied** for a *resource* R_2 if and only if there are at least N different instances of R that are related to R_2 by P.
$\forall C \in \mathcal{C}_{min-card} \quad \forall R_2 \in \text{rdfs:Resource}(satisfiedFor(C,R_2) \Leftrightarrow C = \text{<P,R,N>} \wedge$
$\exists \mathcal{R}(|\mathcal{R}| \geq N \wedge \forall R_3 \in \mathcal{R}(P(R_2,R_3) \wedge R_3 \in R))$

Following [224], a **minimum-cardinality restriction** is a *restriction* with a **minimum-cardinality constraint**. *Minimum-cardinality constraints* are typically used to describe *concrete components* whose instances have structures that are obligatorily dependent on other *concrete components*. For example, every instance of `cro:StructuralComponent` includes at least one instance of `cro:SpatialComponent`.

Following [224], an **exact-cardinality constraint** C=<P,R,N> is a cardinality constraint that **is satisfied** for a *resource* R_2 if and only if there are exactly N different instances of R that are related to R_2 by P.
$\forall C \in \mathcal{C}_{exact-card} \quad \forall R_2 \in \text{rdfs:Resource}(satisfiedFor(C,R_2) \Leftrightarrow C = \text{<P,R>} \wedge$
$\exists \mathcal{R}(|\mathcal{R}| = N \wedge \forall R_3 \in \mathcal{R}(P(R_2,R_3) \wedge R_3 \in R))$

Following [224], an **exact-cardinality restriction** is a *restriction* with an **exact-cardinality constraint**. *Exact-cardinality restrictions* are typically used to describe *concrete components* whose instances have features dependent on other *concrete components*. For example, every instance of `cro:TextureMaterial` has exactly one value of the `cro:path` *property*; every `cro:TwoSidedObject` has exactly two instances of `cro:Material` assigned.

Following [224], a **maximum-cardinality constraint** C=<P,R,N> is a cardinality constraint that **is satisfied** for a *resource* R_2 if and only if there are at most N different instances of R that are related to R_2 by P.
$\forall C \in \mathcal{C}_{max-card} \quad \forall R_2 \in \text{rdfs:Resource}(satisfiedFor(C,R_2) \Leftrightarrow C = \text{<P,R>} \wedge$
$\exists \mathcal{R}(|\mathcal{R}| \leq N \wedge \forall R_3 \in \mathcal{R}(P(R_2,R_3) \wedge R_3 \in R))$

Following [224], a **maximum-cardinality restriction** is a restriction with a **maximum-cardinality constraint**. *Maximum-cardinality restrictions* are typically used to describe *concrete components* whose instances have features that are optionally dependent on other *concrete components*. For example, instances of *concrete components* whose materials have no `cro:specularColor` specified, have no specular highlight. In contrast to the other restrictions, *exact-cardinality restrictions* and *maximum-cardinality restrictions* require the *closed world assumption*, in which a statement that is not specified is considered false.

Multiple *concrete components* can be combined using *object properties* to enable representation of complex 3D content.

Definition 26. *A concrete component C_1 is a* **subcomponent** *of a concrete component C_2 if and only if C_2 is a subclass of a restriction with a constraint on a concrete object property and C_1.*

For example, `crr:TreeComponent` that extends `cro:AppearanceComponent`, has multiple *subcomponents* `crr:TreeLevelOfDetail` (that extend `cro:LevelOfDetail`) that determine the presentation of the trees depending on the distance to the observer.

5.5. Conceptual 3D Content Representation

In this section, DSOs and CpRs that may be used in SEMIC are explained. First, an overview of a DSO and a CpR is provided. Second, the formal model of CpRs is described. Third, the elements of DSOs and CpRs are presented in detail.

5.5.1. Overview of Conceptual 3D Content Representation

A CpR is an SCM knowledge base that is an instance of a DSO. A CpR consists of domain-specific individuals, which are described by domain-specific properties. Domain-specific classes and properties are used to represent 3D content at an arbitrarily high (arbitrarily chosen) level of abstraction (Chapter 4/Requirement 2). Domain-specific classes and properties are abstract in the sense of their final presentation, as—in general—they can be presented in various manners (e.g., 2D graphics, 3D models, aural or haptic elements). Domain-specific classes and properties typically do not cover aspects related to

3D content, or such aspects are not directly indicated. For instance, a car does not need to be represented by a particular 3D shape, though it may be considered as such in terms of its final presentation by belonging to a subclass of concrete components that are 3D meshes. Since SEMIC is independent of DSOs that can be used for modeling 3D content, neither the creation nor the selection of DSOs is addressed in the dissertation. To enable 3D representation of domain-specific individuals and properties, DSOs are mapped to CrRs by RMs.

5.5.2. Formal Model of Conceptual 3D Content Representation

Definition 27. *A **Conceptual 3D Content Representation (CpR)** is such an SCM knowledge base that is an instance of a DSO.*

Definition 28. *A **domain-specific class** is such an SCM class that is defined in a DSO.*

Definition 29. *A **domain-specific property** is such an SCM property that is defined in a DSO.*

Definition 30. *A **domain-specific individual** is such an SCM individual that is an instance of a domain-specific class.*

Different CpRs can be created using *SCM classes* and *SCM properties defined in* a DSO. A CpR is a *representation* of some specific *3D content*, which may be a *3D object* or a *3D scene*. Examples of CpRs are: a model of a city with buildings, streets and cars; a game with different characters; a virtual museum with rooms, exhibitions, artifacts and visitors.

A CpR comprises *rules* on *domain-specific classes*, *domain-specific individuals* and *domain-specific properties*. Examples of *statements* on *domain-specific individuals* are: the earth (subject) goes around (SCM property) the sun (object), an artifact (subject) stands on (SCM property) a monument (object), a car (subject) runs on (SCM property) a road (object), etc.

5.6. Mapping 3D Content Representations

In this section, the MO and RM are explained. First, an overview of the MO and RM is provided. Second, the formal model of representation mapping is described. Third, the elements of the MO and RMs are presented in detail.

5.6.1. Overview of Representation Mapping

An RM is an SCM ontology that is an instance of the MO (proposed in [134]), which maps a DSO to a CrR. The goal of RM is to make domain-specific classes and properties (included in the DSO) presentable by associating them with concrete components and properties (included in the CrR). SEMIC does not restrict the acceptable kinds or domains of DSOs, thus different DSOs may be used to create 3D contents of different types, for different domains and applications. As the cases and contexts of use are already well-defined, domain-specific concepts may be mapped to concrete components by RMs—similarly to encapsulating low-level functions behind high-level objects' interfaces in object-oriented programming.

Mapping domain-specific classes and properties used in CpRs to particular concrete components and properties of a CrR improves modeling efficiency and reusability of the classes and properties in contrast to defining individual 3D representations for particular domain-specific individuals.

5.6.2. Mapping Concepts

In this section, the mapping concepts are discussed. The creation of particular mapping concepts is explained in the discussion of SCCM in Section 6.3.

Presentable Object Classes

Presentable objects (POs), which are instances of *presentable object classes* (POCs), are the primary SCM individuals of a CpR. Every SCM class defined in a DSO, whose instances represent independent entities in the created 3D content (having own geometry, structure and appearance) is specified as a subclass of a POC. Every POC determines some SCM properties that are inherent to its POs—form the POs and give a sense of them in the selected presentation modality, e.g., the color map of a picture, the geometry of a 3D shape, the structure of a complex object, the sampling frequency of a sound, etc. A POC is a subclass of concrete components, at least one of which inherits from `cro:GeometricalComponent`, `cro:StructuralComponent` or `cro:AppearanceComponent`. Hence, a mapped domain-specific class is associated with the SCM properties associated with the concrete components. Examples of POs are artifacts in a virtual museum exhibition, avatars in an RPG game, UI controls in a visual interface and sounds in an aural interface. POs in a CpR may be described by the other mapping concepts defined in the RM—DCs, DIs and MPs. In contrast to POs, the remaining mapping concepts describe POs and reflect aspects of 3D content that have no independent representations in the final 3D scene, e.g., a texture or an animation do not exists independently in a 3D scene, but are always assigned to some objects.

Descriptive Classes

A *descriptive class* (DC) is a subclass of concrete components and a superclass of a domain-specific class. Hence, SCM individuals that belong to the domain-specific class are described by the concrete components from which the DC inherits. For example, by using a DC, the `dso:OakWood` class is a subclass of a concrete component inheriting from `cro:Material` that is described by an appropriate texture. The assignment of `dso:OakWood` to a domain-specific individual that describes the appearance of furniture results in the assignment of the concrete component (with the texture) to the furniture. Another example—`dso:RotatingMotion` is a subclass of a concrete component inheriting from `cro:InterpolatedAnimation` that has a concrete subcomponent inheriting from `cro:PositionInterpolator`. The assignment of `dso:RotatingMotion` to a domain-specific individual that describes the behavior of planets results in the rotation of the planets.

DCs are not used to represent independent 3D objects or 3D scenes. Instead, they are used to enrich POs with additional concrete components that cannot be defined inherently to a whole POC, but need to be assigned selectively only to some POs of a POC. For example, all boxes (a POC) and doors (another POC) in a magazine are made of wood (a DC); an SCM class of interactive rotating objects consists of only selected exhibits (POs) that rotate after being touched.

Mapping Properties

POs can be described by *mapping properties* (MPs). An MP is a superproperty of a domain-specific property. In addition, on every MP, has-value restrictions are specified. The restrictions associate distinguished values of the MP with particular DCs. Hence, the SCM individuals with a particular value of the MP belong to the DC and thereby, they are instances of the concrete components that are superclasses of the DC. A *mapping data property* (MDP) is an MP whose range is a class of literal values (floats, integers, strings, etc.), e.g., a pot is made of ’clay’, the size of a monument is ’large’, the frequency of a pendulum is ’high’. A *mapping object property* (MOP) is an MP whose range is a *descriptive individual class* (DIC), e.g., a descriptor of a table specifies the type of the tableware used and the number of place settings. In general, different MPs and DCs that are assigned to a PO may determine common concrete components of the PO. For example, the color of a waxen object depends on both its temperature and the dye used; the appearance of the leafs of a tree depends on its species, the season and diseases.

Descriptive Individual Classes

In addition to MPs, POs may be described by DIs, which are instances of DICs. A DI is an SCM individual that is related to the described PO by an MOP. Although a DI may be described by numerous MDPs and MOPs, the concrete properties assigned to a DI (through MPs) are applied to the described PO—the DI only carriers the properties. For example, furniture can be made of different types of wood, each of which is described by a few SCM properties such as age, species and shininess.

Relations

DIs may be related to POs by MOPs or two POs may be related by a MOP. In such cases, DIs/MOPs are used to represent *relations* (RLs) between the POs. A relation combines at least two POs by dependencies on some MDPs or MOPs of related POs. In every relation, at least one PO affects other POs and at least one PO is affected by other POs. Moreover, each PO is either affecting or affected. An n-ary RL is built on a DI that is related to multiple POs by MOPs. A binary RL is built on an MOP that links two POs.

For example, a relation that determines the relative position of some POs specifies their relative orientations and distances between them; an object that is placed on a table takes table’s height to calculate one of the position coordinates and does not change any SCM properties of the table.

5.6.3. Formal Model of Representation Mapping

Instances of the MO are RMs. An RM enables mapping between a DSO and a CrR.

Definition 31. *A* ***Representation Mapping (RM)*** *is such an SCM ontology that is an instance of the MO.*

RMs are created using *mapping concepts*. **Mapping concepts** are *SCM classes* and *SCM properties* used to map domain-specific classes and properties to concrete components and concrete properties. The following mapping concepts are distinguished in the MO: *presentable objects classes* (POCs), *descriptive classes* (DCs), *mapping properties* (MPs) and *descriptive individual classes* (DICs). Since the DSO is mapped to a CrR, all *domain-specific individuals* used for modeling *3D content* are instances of *descriptive classes*.

Definition 32. *A* ***descriptive class (DC)*** *is such an SCM class that is:*

— *a subclass of the Root Descriptive Class defined in the MO, and*
— *a subclass of concrete components, and*
— *a superclass of a domain-specific class.*

A specific kind of DC is *presentable object class*.

Definition 33. *A* ***presentable object class (POC)*** *is such a DC that is a subclass of the Root Presentable Object Class defined in the MO, and a subclass of:*

— `cro:GeometricalComponent` *or*

— *`cro:StructuralComponent` or*
— *`cro:AppearanceComponent`.*

Definition 34. *A **presentable object (PO)** is an instance of a POC.*

Hence, POs have presentational properties related to geometry, structure and appearance. *Domain-specific individuals* are described by *mapping properties* and by other *domain-specific individuals* (*descriptive individuals*).

Definition 35. *A **mapping property (MP)** is such a domain-specific property that:*

— *it is a subproperty of the Root Mapping Property defined in the MO, and*
— *has-value restrictions with has-value constraints on MP and its particular values exist and they are equivalent to some DCs, and*
— *the domain of MP is a DC.*

Definition 36. *A **mapping data property (MDP)** is such a mapping property that is an SCM data property.*

Definition 37. *A **mapping object property (MOP)** is such a mapping property that is an SCM object property.*

MOPs are used to link different *domain-specific individuals*.

Definition 38. *A **descriptive individual class (DIC)** is such an SCM class that is:*

— *a subclass of the Root Descriptive Individual Class defined in the MO, and*
— *a subclass of all-values-from restrictions, and*
— *a superclass of a domain-specific class.*

Definition 39. *A **descriptive individual (DI)** is an instance of a DIC.*

A specific kind of DIs and MOPs are *relations* between PO.

Definition 40. *A **relation (RL)** between POs is a subclass of the Root Relation Class defined in the MO, and:*

— *a MOP whose domain and range are POCs, or*
— *a DIC that is a subclass of restrictions on at least two different POCs.*

5.7. Transformation of 3D Content Representations

In this section, the TO, TKBs and TBs are explained. First, an overview of the TO, a TKB and a TB is provided. Second, the relevant formal definitions are provided. Third, the particular elements of a TKB and a TB are described in detail. The use of a TKB for transforming 3D content representations is explained in detail in the discussion of SCCM in Section 6.7.

5.7.1. Overview of Transformation

Semantic 3D content representations are independent of particular 3D content representation formats and languages. They may be transformed to be presentable on different content presentation platforms. TO enables description of the transformation of *Platform-independent 3D Content Representations*

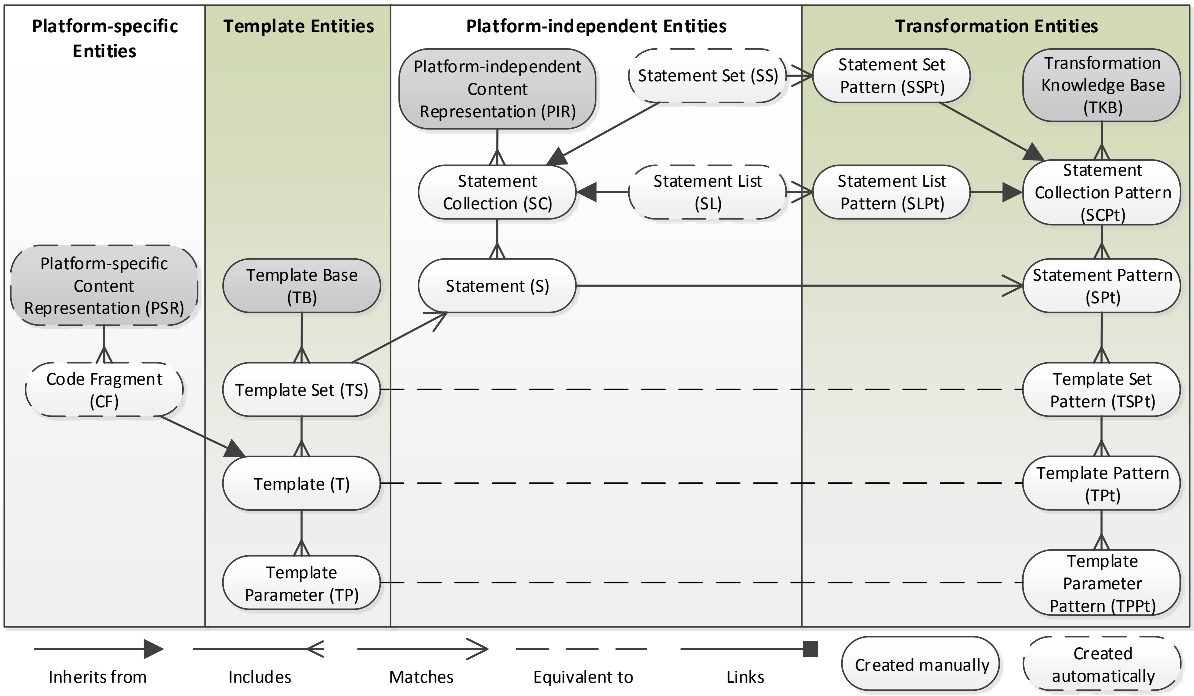

Platform-specific Entities
Template Entities
Platform-independent Entities
Transformation Entities
Platform-independent Content Representation (PIR)
Statement Set (SS)
Statement Set Pattern (SSPt)
Transformation Knowledge Base (TKB)
Statement Collection (SC)
Statement List (SL)
Statement List Pattern (SLPt)
Statement Collection Pattern (SCPt)
Platform-specific Content Representation (PSR)
Template Base (TB)
Statement (S)
Statement Pattern (SPt)
Code Fragment (CF)
Template Set (TS)
Template Set Pattern (TSPt)
Template (T)
Template Pattern (TPt)
Template Parameter (TP)
Template Parameter Pattern (TPPt)
Inherits from
Includes
Matches
Equivalent to
Links
Created manually
Created automatically

Figure 5.3: The elements of a *Transformation Knowledge Base* (TKB) and a *Template Base* (TB)

(PIRs) to *Platform-specific 3D Content Representations* (PSRs) [140, 141], which can be presented on specific platforms (Chapter 4/Requirement 6). A PIR is an SCM knowledge base that is an instance of a CrR. PIRs are automatically generated by the *Expanding Algorithm* of SCCM (cf. Section 6.5). A PIR consists of statements on concrete properties and instances of concrete components (defined in the CrR). PIRs are automatically transformed to PSRs by the *Transformation Algorithm* of SCCM (cf. Section 6.7).

A transformation of PIRs to PSRs encoded in a particular 3D content representation language is described by a TKB and a TB. While the transformation algorithm is platform-independent, TKBs and TBs are platform-dependent. A TKB is a set of rules describing the transformation for a particular 3D content representation language. A TKB is used with a TB, which includes parametrized templates of code in the 3D content representation language. A new TKB and TB is created once every time a new 3D content presentation platform (a combination of hardware and software) that uses a new 3D content representation language is introduced to the system. When a new TB and a new TKB are added, they may be used to transform PIRs to PSRs that may be presented on the new platform. The generated PSRs comprise templates of the TB combined and filled according to the TKB. PSRs are visualized using diverse 3D content representation languages (e.g., X3D, Java), programming libraries (e.g., Java3D, Away3D) and game engines (e.g., Unity, Unreal). The aforementioned concepts are described in detail in Sections 5.7.3 and 5.7.2. The elements of a TKB, a TB, a PIR and a PSR as well as relations between them are depicted in Figure 5.3.

5.7.2. Transformation Concepts

The particular elements related to transformation of PIRs to PSRs are discussed in the following sections.

Platform-independent Content Representations

A PIR is an SCM knowledge base that is an instance of a CrR, thus it is presentation platform agnostic. However, PIRs can be presented on particular platforms when transformed with appropriate TKBs. A PIR includes statements on concrete properties and instances of concrete components. Both the properties and the components reflect different aspects of 3D content, such as geometry, structure, space, appearance, animation and behavior.

A statement is the basic entity of a PIR in terms of transformation. During the transformation, statements are assembled into *statement collections* (SCs). Assembling statements into SCs enables the elementary operations (setting TPs, nesting Ts and ordering Ts) on Ts that are associated with the statements. Two kinds of SCs are distinguished: *statement sets* (SSs) and *statement lists* (SLs), which are built with respect to SSPts and SLPts, respectively.

Transformation Knowledge Base

A TKB describes the transformation of PIRs to PSRs. An individual TKB is created for a particular presentation platform or a group of presentation platforms that use a common 3D content representation language or different languages that have equivalent structures of documents. Since the transformation is generic and based on elementary operations on semantic statements and code templates, TKBs for different—both declarative (e.g., VRML, X3D) and imperative (e.g., ActionScript, Java)—3D content representation languages may be built.

The primary entity of a TKB is a *statement pattern* (SPt). Every SPt is a statement that is matched by a group of statements that may potentially occur in a PIR. Matching an SPt by a group of statements is enabled by semantic generalization. A generalization may pertain to the subject, the SCM property or the object of statements. For instance, a possible generalization of S_1 `[object scm:color "red"]` in terms of SCM property, is S_2 `[object scm:appearanceProperty "red"]`, while a possible generalization of S_3 `[object rdf:type scm:Mesh3D]` in terms of object, is S_4 `[object rdf:type scm:GeometricalComponent]`.

SPts include entities that determine the structure of the resulting PSR. An SPt may include a *template set pattern* (TSPt) with a number of *template patterns* (TPts), each of which may contain a number of *template parameter patterns* (TPPts). These concepts correspond to the entities of parametrized code, which are used in transformation: *template sets* (TSs), *templates* (Ts) and *template parameters* (TPs), respectively. However, the concepts do not explicitly indicate the entities of parametrized code. In a TKB, the level of generality of SPts may be high to cover a number of statements in a PIR. For instance, the SPt `[?subject scm:dataProperty ?value]` may cover the statements `[?subject scm:intensity "10"]` and `[?subject scm:color "red"]`. In a PSR, these statements are represented by different Ts. Hence, the selection of a T is done for a particular statement. However, the general structure of both Ts may be known in advance and specified at the semantically generalized level in a TKB. For instance, the aforementioned Ts may be specified as `[$object.intensity = $data]` and `[$object.color = $data]`. Both the statements use the `object` and `data` TPs.

The following elementary operations are performed on Ts to transform PIRs to PSRs:

(1) setting common values of TPs,
(2) nesting a T into another T,
(3) ordering Ts.

The operations are enabled by *statement collection patterns* (SCPts): operations (1) and (2)—by *statement set patterns* (SSPts), while operation (3)—by *statement list patterns* (SLPts). Every SCPt includes multiple SPts. As an SPt matches a single statement, an SCPt matches a group of statements in a PIR.

For instance, the SLPt `[?subject ?property ?value. ?subject rdf:type ?type.]` matches the pair of statements `[?light scm:intensity "10". ?light rdf:type scm:DirectionalLight.]`. The common subject (`light`) and the reverse order needs to be set for an imperative 3D content representation language (e.g., ActionScript), while the T of the first statement needs to be nested into the T of the second statement for a declarative 3D content representation language (e.g., X3D). The resulting imperative code is `[DirectionalLight light = new DirectionalLight(); light.intensity = 10;]`. To enable linking TPs and nesting Ts between different statements of an SCPt, appropriate statements are specified for TPPts and TPts (which reflect TPs and Ts) in an SCPt.

Template Bases

A TB is a set of parametrized code templates that may be used to create PSRs on the basis of PIRs. The primary entity of a TB is a *template set* (TS), which consists of *templates* (Ts), which are parametrized fragments of code—may include a number of *template parameters* (TPs). A TB conforms to a 3D content representation language (e.g., X3D, Java), possibly with programming libraries (e.g., Java3D, Away3D). Every statement in a PIR may be matched by a TS. Matching is dynamic and based on the actual *signature* of the statement, which is a triple `<class of the subject, property, class of the object>`. For every statement, the TS whose *signature* matches the statement is selected. Since the selection of a TS requires a particular statement, matching between TSs and statements cannot be given in a TKB in SPts. For instance, both the statements: `[?obj1 rdf:type cro:DirectionalLightSource]` and `[?obj2 rdf:type cro:SpotLightSource]` match the SPt `[?obj rdf:type cro:LightSource]`. However, the first statement is matched to an X3D TS including the T `<DirectionalLight ...>`, while the second statement—to a TS including the T `<SpotLight ...>`.

Every TS may include a number of Ts that need to be individually processed, e.g., injected into different TPs of another T. The processing of particular Ts, which are included is a TS, is specified in the TKB. For instance, for a presentation platform that uses an imperative 3D content representation language without navigation implemented, it may be necessary to inject a T implementing proper functions next to the `main` function and to inject another T invoking the functions within the `main` function.

A TB and a TKB are created once every time a new presentation platform using a new content representation language is added to the system. When a new TB and a new TKB are introduced, they may be used to generate various PSRs that are presentable on the new platform.

Platform-specific 3D Content Representations

A PSR is a collection (a set or a sequence) of *code fragments* (CFs) encoded in a 3D content representation language. A PSR is generated automatically by the transformation algorithm (cf. Section 6.7) upon Ts by setting their TPs. Once created, a PSR may be presented on the selected 3D content presentation platform.

5.7.3. Formal Model of Transformation

The TO is used to create *Transformation Knowledge Bases* (TKBs).

Definition 41. *A **Transformation Knowledge Base (TKB)** is such an SCM ontology that is an instance of the TO.*

A TKB describes the transformation of *Platform-independent 3D Content Representations*, which are *semantic 3D content representations*, to *Platform-specific 3D Content Representations*, which are *3D content representations* encoded in a **3D content representation language**, which is a programming language used to create 3D content representations. Different *3D content representation languages* may be used in SEMIC, e.g., Java, ActionScript, VRML and X3D. The *3D content representation languages* used determine *3D content presentation platforms*, which are combinations of hardware and software that enable presentation of 3D content. Examples of *3D content presentation platforms* are: a smartphone equipped with an Internet browser that supports WebGL, a desktop equipped with an X3D browser, a CAVE system equipped with specific software for 3D content presentation, a powerwall equipped with software enabling stereoscopic rendering, etc.

Definition 42. *A **Platform-independent 3D Content Representation (PIR)** is such an SCM knowledge base that is:*

— *an instance of a DSO, and*
— *an instance of a CpR, and*
— *an instance of a CrR.*

A PIR consists of *statement collections*.

Definition 43. *A **statement collection (SC)** is a collection of statements.*

Two kinds of *statement collections* are distinguished.

Definition 44. *A **statement set (SS)** is a set of statements.*

Definition 45. *A **statement list (SL)** is a sequence of statements.*

In an SL, the order of *statements* matters, as opposed to an SS. Respecting the order of *statements* is mostly required in the case of TBs encoded in imperative *3D content representation languages*, whereas it is not important in the case of TBs encoded in declarative *3D content representation languages*. An example of an SC is: `[?light rdf:type scm:DirectionalLight. ?light scm:intensity "10".]`. If the SC is an SL, the first *statement* must be processed before the second *statement* (first, the object must be created), whereas if the SC is an SS, the order of processing the *statements* does not matter.

A transformation of a PIR to a PSR that is described by a TKB uses a *Template Base* (TB), which consists of parametrized code *templates*. A **template (T)** is a parametrized document encoded in a 3D content representation language. A **template parameter (TP)** of template T is a named part of T. The name of a TP is unique within the T. TPs are used to nest *templates* and to embed values in *templates*. For example, in the following T (Listing 5.1):

Listing 5.1: A template with template parameters

```
<Appearance DEF='DefaultAppearance'>
      <Material diffuseColor='$Color-Value'/>
      $ImageTexture-TemplateParameter
</Appearance>
```

TPs are used to fill the T with a diffuse color as well as with another T representing an image texture.

Definition 46. *A template parameter TP of template T* ***is filled*** *with a value V (or with a template T_2) if and only if V (or T_2) occurs in T instead of TP.*

Definition 47. *A template* ***is filled*** *if and only if all its template parameters are filled.*

For example, the filled T from Listing 5.1 is presented in Listing 5.2.

Listing 5.2: A filled template

```
<Appearance DEF='DefaultAppearance'>
      <Material diffuseColor='1 0.4 0.9'/>
      <ImageTexture DEF='texture1' url=' ".../texture1.png" '/>
</Appearance>
```

Definition 48. *A* ***template set (TS)*** *is a set of templates.*

Definition 49. *A* ***template base (TB)*** *is a set of template sets.*

Definition 50. *A* ***code fragment (CF)*** *is a filled template.*

For example, a CF responsible for specifying a normal interpolation in X3D includes the element with some attributes:

```
<CoordinateInterpolator2D DEF='CoordID' key='...' keyValue='...'/>
```

A **Platform-specific 3D Content Representation (PSR)** is a 3D content representation encoded in a 3D content representation language.

Definition 51. *A PSR is a collection (a set or a sequence) of code fragments.*

Definition 52. *A* ***template parameter pattern (TPPt)*** *is an instance of the Template Parameter Pattern Class defined in the TO.*

In a transformation, the name of each TPPt is equal to the name of a TP of a T. A TPPt, which is included in a TPt, is a counterpart to a TP, which is included in a T.

Definition 53. *A* ***template pattern (TPt)*** *is an instance of the Template Pattern Class defined in the TO.*

A TPt represents a set of TPPts. A TPt, which is included in a TSPt, is a counterpart to a T, which is included in a TS.

Definition 54. *A* ***template set pattern (TSPt)*** *is an instance of the Template Set Pattern Class defined in the TO.*

A TSPt represents a set of template patterns. A TSPt is a counterpart to a TS, which is included in a TB. The concepts defined in the TO and used to create TKBs correspond to the concepts distinguished within PIRs.

Definition 55. *The Statement Pattern class is a class of pairs <S,TSPt>, where:*

— *S is a statement,*
— *TSPt is a template set pattern.*

Definition 56. *A* ***statement pattern (SPt)*** *is an instance of the Statement Pattern Class defined in the TO.*

Let $\mathcal{SP}t$ denote the set of **all possible statement patterns**.

Definition 57. *Statement S <S_1,P_1,O_1>* ***matches*** *statement pattern SPt < <S_2,P_2,O_2>,TSPt> if and only if:*

— *S_1 is equal to S_2 or S_1 is an instance of S_2 or S_1 is a subclass of S_2, and*
— *P_1 is a subproperty of P_2, and*
— *O_1 is equal to O_2 or O_1 is an instance of O_2 or O_1 is a subclass of O_2.*

$$\forall S \in \mathcal{S} \quad \forall SPt \in \mathcal{SPt}(matches(S,SPt) \Leftrightarrow S =< S_1,P_1,O_1 > \wedge SPt =<< S_2,P_2,O_2 >,TSPt > \wedge (S_1 = S_2 \vee instanceOf(S_1,S_2) \vee subclass(S_1,S_2)) \wedge subproperty(P_1,P_2) \wedge (O_1 = O_2 \vee instanceOf(O_1,O_2) \vee subclass(O_1,O_2)))$$

A *statement* S matches a *statement pattern* SPt if there are specific relations between their *subjects*, *SCM properties* and *objects*. If the *subject/object* of S is an *SCM individual*, it should be equal to the *subject/object* of SPt (S matches SPt that describes the *SCM individual*), or it should be an instance of the *subject/object* of SPt (S matches SPt that describes the class of SCM individuals). If the *subject/object* of S is an *SCM class*, it should be a subclass of the *subject/object* of SPt (S matches SPt that is a generalization of S).

Definition 58. *A* ***statement collection pattern (SCPt)*** *is an instance of the Statement Collection Pattern Class defined in the TO.*

An SCPt represents a collection of SPts. Two kinds of SCPts are distinguished (like in the case of SCs).

Definition 59. *A* ***statement set pattern (SSPt)*** *is a set of statement patterns.*

Definition 60. *A* ***statement list pattern (SLPt)*** *is a sequence of statement patterns.*

As a *statement* may match SPts, an SC may match SCPts: an SS may match SSPts and an SL may match SLPts.

6. The SEMIC Semantic 3D Content Creation Method

In this chapter, the SEMIC *Semantic 3D Content Creation Method* (SCCM) is described [137, 139, 141, 238]. SCCM offers an important advantage over the previous approaches by enabling flexible creation of interactive 3D content using domain-specific ontologies. SCCM supports separation of concerns in content creation between different modeling users, on-demand 3D content customization by different content consumers (users and applications), and discovery of tacit knowledge from 3D content representations. First, the outline of SCCM is provided, then the subsequent steps of SCCM, which are performed by different users and algorithms, are discussed.

6.1. Outline of the SCCM Method

SCCM enables flexible declarative creation of 3D content at an arbitrarily chosen level of abstraction by using SCM and CCPs (Chapter 4/Requirements 1 and 2). In SCCM, 3D content is created within a sequence of steps that correspond to different levels of semantic abstraction of the created content. The following steps are distinguished in SCCM (Figure 6.1):

— Step 1: designing a concrete 3D content representation,
— Step 2: mapping the concrete 3D content representation to a domain-specific ontology,
— Step 3: designing a conceptual 3D content representation,
— Step 4: expanding the conceptual 3D content representation into a 3D meta-scene,
— Step 5: customizing the 3D meta-scene,
— Step 6: generating platform-specific 3D content representations.

The steps use different SCM ontologies and produce 3D content representations, which are compliant with the SCM ontologies. 3D content creation based on SCCM may be performed using the SO-SEMIC environment (cf. Chapter 7) or separate tools for modeling 3D content (e.g., Blender [25] or 3ds Max [82]) and semantic descriptions (e.g., Protégé [60]). SCCM supports the division of responsibilities in 3D content creation between different users and software procedures (Chapter 4/Requirement 4). The succeeding steps of SCCM depend on the results of their preceding steps. Steps 1-3 are performed manually by users with different skills in modeling of 3D content—a modeler, a developer and a domain expert. Steps 4 and 6 are performed automatically by specific SCCM software procedures. Step 5 is performed in part manually by a content user and in part automatically by a software procedure. The presented 6 steps precede the final 3D content presentation to a user, which may be performed using various 3D content browsers and presentation tools, such as Bitmanagement BS Contact [24] and Cortona3D [44]. The 6 steps of SCCM are briefly described below.

Step 1 provides concrete components of 3D content to enable representation of domain-specific concepts (classes and properties) that will be further used in Step 3. The result of Step 1 is a CrR, which is an SCM knowledge base compliant with the CRO. Concrete components included in a CrR are grouped into several partly dependent layers that are specific to 3D content—geometry layer (e.g., shapes, meshes, planes), structure layer (e.g., groups of objects, size, position, orientation), appearance layer

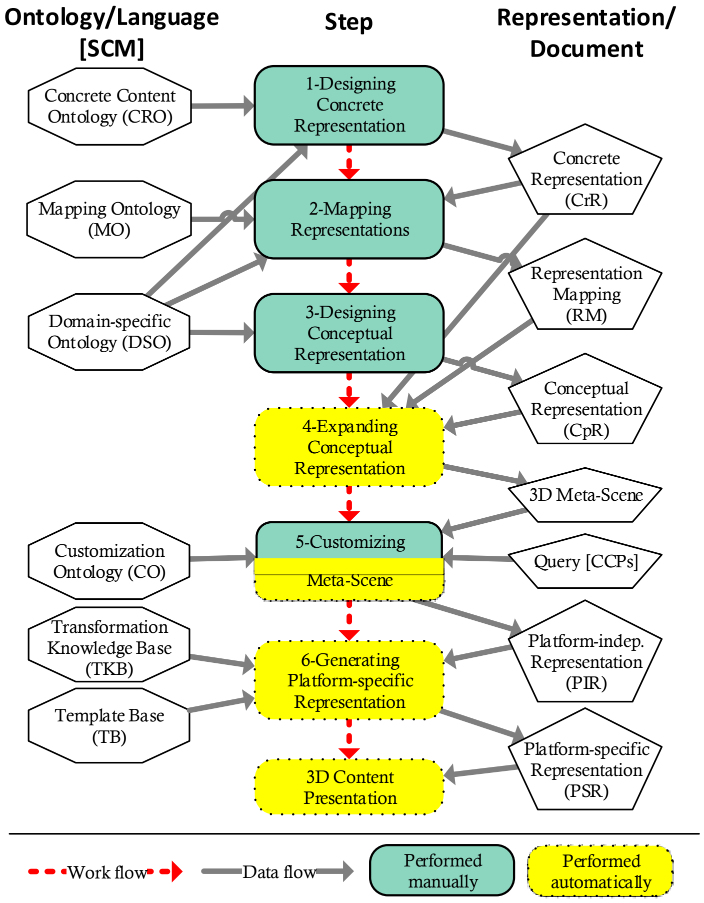

Ontology/Language [SCM]
Step
Representation/ Document
Concrete Content Ontology (CRO)
1-Designing Concrete Representation
Concrete Representation (CrR)
Mapping Ontology (MO)
2-Mapping Representations
Representation Mapping (RM)
Domain-specific Ontology (DSO)
3-Designing Conceptual Representation
Conceptual Representation (CpR)
4-Expanding Conceptual Representation
3D Meta-Scene
Customization Ontology (CO)
5-Customizing
Meta-Scene
Query [CCPs]
Transformation Knowledge Base (TKB)
Platform-indep. Representation (PIR)
6-Generating Platform-specific Representation
Template Base (TB)
Platform-specific Representation (PSR)
3D Content Presentation
Work flow
Data flow
Performed manually
Performed automatically

Figure 6.1: Creation of 3D content based on the *Semantic 3D Content Creation Method* (SCCM)

(e.g., textures, materials, light sources), scene layer (e.g., viewpoints, navigation), animation layer (e.g., interpolators) and behavior layer (e.g., mouse and keyboard events). This step is typically performed by a developer with expertise in 3D modeling, who is equipped with specific tools, e.g., 2D and 3D graphical editors for creating textures, meshes, etc.

Step 2 enables creation of 3D representations of conceptually modeled content (CpRs that will be created in Step 3). The result of this step is an RM, which is compliant with the MO. Mapping is performed by a developer once for a particular DSO and a CrR. An RM enables the reuse of concrete components and properties for forming 3D representations of various domain-specific individuals, which conform to the domain-specific ontology used.

Step 3 enables creation of 3D content at an arbitrarily chosen level of abstraction that is determined by the DSO used. The resulting CpR is compliant with the DSO. This step can be performed multiple times for a particular DSO, a CrR and an RM. This step requires the knowledge of particular domain-specific concepts but not computer graphics concepts. Thus, it is typically performed by a domain expert, who is equipped with a semantic modeling tool (possibly domain specific).

Step 4 combines the CpR with the CrR using the RM. It produces an overall semantic content representation—a 3D meta-scene—which is an SCM knowledge base representing the content at both concrete and conceptual levels of abstraction, i.e. how particular domain-specific individuals are represented by particular concrete components and properties. This step is completed automatically by the *expanding algorithm*.

Step 5 enables customization of content by selection of particular domain-specific individuals of the meta-scene that are to be presented and specification which features and behavior of the SCM individuals should be presented. Content customization may be accomplished at different levels of abstraction (both concrete and conceptual), as permitted by the meta-scene. This step may be performed on demand, by different content consumers (users and applications), which independently execute queries to meta-scenes. Queries are built according to CCPs. Queries specify requirements for 3D content customization. The result of this step is a PIR. A PIR is a customized content representation, which satisfies the requirements. The resulting PIRs reflect both explicit and implicit (tacit) knowledge included in the meta-scenes and the queries. The customization of a meta-scene on the basis of a query is completed by the *customization algorithm*.

Step 6 is completed automatically using the *transformation algorithm* and a TKB that links concrete components to their corresponding counterparts included in a TB. The transformation can cover a wide range of target presentation platforms based on either declarative (e.g., VRML, X3D and XML3D) or imperative (e.g., Java, ActionScript and JavaScript) content representation languages.

The following sections describe the subsequent steps of the modeling process, along with an example, in which different 3D content representations of a virtual museum of agriculture are created by different users and algorithms (Listings 6.1-6.9).

6.2. Step 1: Designing a Concrete 3D Content Representation

In this section, the concept and an example of designing CrRs in Step 1 of SCCM are discussed.

6.2.1. The Concept of Designing Concrete Representations

Designing a CrR provides concrete components of 3D content, which are a foundation for the representation of domain-specific classes and properties, which will be constructed in Step 3 (Chapter 4/Requirement 2). A CrR is an SCM knowledge base compliant with the CRO (cf. Section 5.4), which represents 3D content at the concrete level of abstraction. Concrete components, which are the elements of a CrR, are specific to 3D content. Examples of concrete components are meshes, materials, viewpoints, events and complex structural objects. A CrR may be created at an arbitrary layer of the CRO, including concrete components based on SCM classes and SCM properties defined in this layer and its lower layers, e.g., designing a mesh is related to the geometry layer, while designing a motion trajectory is related to the animation layer. In general, formation of concrete components is complex, and it may require the use of additional specific hardware or software tools. For instance, the creation of a 3D mesh requires the use of a 3D scanner or a 3D modeling tool, while drawing a texture requires a 2D graphical editor. CrRs created in this step represent neither particular coherent 3D scenes nor particular compositions of 3D objects, but they include (possibly independent) templates of reusable content elements that may be flexibly composed into representations of complex 3D objects and 3D scenes.

In most cases, concrete components need to be designed with regards to the domain-specific classes and properties that are to be presented, e.g., a particular 3D mesh represents specific car models, a particular texture represents clay surface, etc. Hence, concrete components need to be created in collaboration with domain experts, who will further use the components in Step 3.

As described in Section 5.4, every concrete component is a subclass of a CRO class and a subclass of restrictions on concrete properties. Concrete properties may have literal values (concrete data properties) or indicate subcomponents (concrete object properties), which are also described by concrete properties. Literal values of concrete properties may be directly used (interpreted) in the content representation (e.g., when representing coordinates or color maps) or they may indicate external data sets (e.g., paths to documents including meshes or images). The specification of literal values, which in general, may be complex, is done using the SO-SEMIC environment (presented in detail in Chapter 7) or additional software (e.g., a 3D modeling tool) or hardware (e.g., a 3D scanner) for creating visual (2D or 3D), haptic or aural elements, etc. This step is typically performed by a modeler with technical skills in 3D modeling with the appropriate tools.

6.2.2. Example of Designing a Concrete Representation

In Step 1 of the considered example, a modeler creates several virtual objects that represent real museum artifacts. First, the modeler uses specific modeling tools to create graphical elements required for low-level 3D content representation—a 3D scanner to capture the geometry of: a granary, a woman statuette, a sower, a smoker, a ring, a seal and a badge, and a 2D graphical tool—to prepare textures for selected 3D models (Figure 6.2). Second, the modeler uses a semantic modeling tool (which may be a plug-in to a modeling package, e.g., Blender [25] or 3ds Max [82]), to create a CrR, which semantically represents the created graphical elements. Listing 6.1 presents an example CrR encoded in the RDF Turtle format [61]. Some concrete components and properties that are not crucial for the example have been skipped. The prefixes used in the listing correspond to different SCM ontologies and content representations. For every 3D model and every texture in the example, concrete components described by concrete properties are generated. The concrete components will be mapped in Step 2 to domain-specific classes and properties used by domain experts in Step 3. In the example, the woman statuette is to be used by domain experts in three different forms: as clay and glassy virtual artifacts (Lines 3-25) and as a painted virtual artifact with the inherent texture mapping (27-34). The other 3D models are to be used in single forms (36-44).

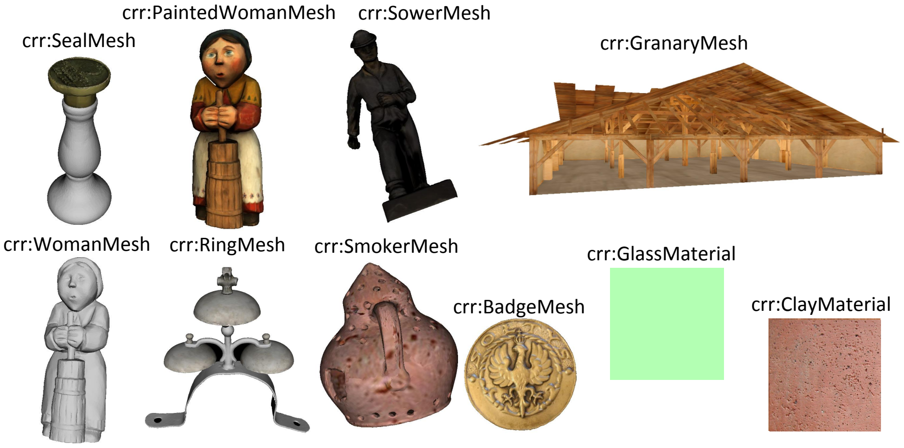


Figure 6.2: An example of concrete 3D content components

Listing 6.1: A concrete 3D content representation (CrR)

```
Prefixes:  Concrete 3D Content Ontology (cro), Concrete 3D Content Representation (crr)

crr:WomanMesh rdf:type owl:Class ;
  rdfs:subClassOf cro:Mesh3D ,
  [ rdf:type owl:Restriction ;
    owl:onProperty cro:meshData ;
    owl:hasValue "woman.obj" ].

crr:ClayMaterial rdf:type owl:Class ;
  rdfs:subClassOf cro:TextureMaterial ,
  [ rdf:type owl:Restriction ;
    owl:onProperty cro:texture ;
    owl:hasValue "clay.png" ] ,
  [ rdf:type owl:Restriction ;
    owl:onProperty cro:transparency ;
    owl:hasValue 0 ].

crr:GlassMaterial rdf:type owl:Class ;
  rdfs:subClassOf cro:ColorMaterial ,
  [ rdf:type owl:Restriction ;
    owl:onProperty cro:color ;
    owl:hasValue "green" ] ,
  [ rdf:type owl:Restriction ;
    owl:onProperty cro:transparency ;
    owl:hasValue 0.7 ].

crr:PaintedWomanMash rdf:type owl:Class ;
  rdfs:subClassOf crr:WomanMesh ,
  [ rdf:type owl:Restriction ;
    owl:onProperty cro:texture ;
    owl:hasValue "statueTexture.png" ] ,
  [ rdf:type owl:Restriction ;
    owl:onProperty cro:textureCoordinates ;
    owl:hasValue "..." ].

{crr:GranaryMesh,..., crr:BadgeMesh}
  rdf:type owl:Class ;
  rdfs:subClassOf cro:Mesh3D ,
  [ rdf:type owl:Restriction ;
    owl:onProperty cro:meshData ;
    owl:hasValue "..." ] ,
  [ rdf:type owl:Restriction ;
    owl:onProperty cro:texture ;
    owl:hasValue "..." ].
```

6.3. Step 2: Mapping the Concrete 3D Content Representation to a Domain-specific Ontology

In this section, the concept and an example of creating RMs in Step 2 of SCCM are discussed.

6.3.1. The Concept of Mapping Representations

Mapping a CrR (created in Step 1) to a DSO enables 3D presentation of domain-specific individuals (created in Step 3) by linking them to concrete components of 3D content included in the CrR. The result of this step is an RM, which comprises mapping concepts that inherit from SCM classes and SCM properties defined in the MO (cf. Section 5.6). Mapping is performed once for a particular DSO and a CrR. Mapping enables the reuse of concrete components for presenting various CpRs, which conform to the DSO. An RM needs to cover all concepts (SCM classes and SCM properties) of the DSO that need to be used in modeling of 3D content.

This step covers operations on the previously designed concrete components, e.g., linking an animation to a domain-specific class, inclusion of meshes within a complex structural object, etc. However, in terms of semantic structures that are created, mapping is more complex and requires more semantic expressiveness than designing a CrR. This step is typically completed by a developer with skills in semantic modeling, using the SO-SEMIC environment or a semantic editor (e.g., Protégé [60]). Mapping is performed on the basis of *mapping patterns* (cf. Section 6.3.2), which are combined into *mapping guidelines* (cf. Section 6.3.3).

Definition 61. *A **mapping pattern** is an elementary sequence of activities that is frequently repeated by modeling users in the creation of different mapping concepts.*

Definition 62. *A **mapping concept** is an SCM class or an SCM property that is used to represent a domain-specific class or a domain-specific property with concrete components or concrete properties.*

Mapping patterns are frequently repeated by a developer when mapping domain-specific classes and properties to concrete components and concrete properties. Multiple mapping patterns are combined into mapping guidelines, which are performed by modeling users to create mapping concepts.

Definition 63. *A **mapping guideline** is a sequence of mapping patterns that is performed by users to create a particular mapping concept.*

6.3.2. Mapping Patterns

Mapping patterns (Figure 6.3) specify means of creating mapping concepts of an RM, which link concrete components (included in a CrR) to domain-specific classes and properties (included in a DSO). Mapping patterns are described in the following sections.

Classification Property

The *classification property* mapping pattern enables reflection of an SCM property P whose values are literal values by a set of SCM classes. For every distinct classification value of P, an individual SCM class is created and it is specified as an equivalent to a has-value restriction on P with the required P value. Consequently, every SCM individual that has a particular classification value of P assigned, belongs to one of the created SCM classes. For instance, objects made of metal, wood and plastic may belong to different SCM classes. Every SCM class created may be further described with different SCM properties, using other mapping patterns.

Multivalued Descriptor

The *multivalued descriptor* mapping pattern enables specification of the desirable SCM data properties for SCM individuals of a common SCM class. To make an SCM class a multivalued descriptor, it needs to be specified as a subclass of the intersection of has-value restrictions. Every has-value restriction indicates a required value for one of the desirable SCM data properties. For instance, every gold object is yellow and reflects light—one restriction specifies color, while the other specifies shininess.

Structural Descriptor

The *structural descriptor* mapping pattern enables creation of a complex structure of SCM classes that are linked by SCM object properties. To make an SCM class a structural descriptor, the class

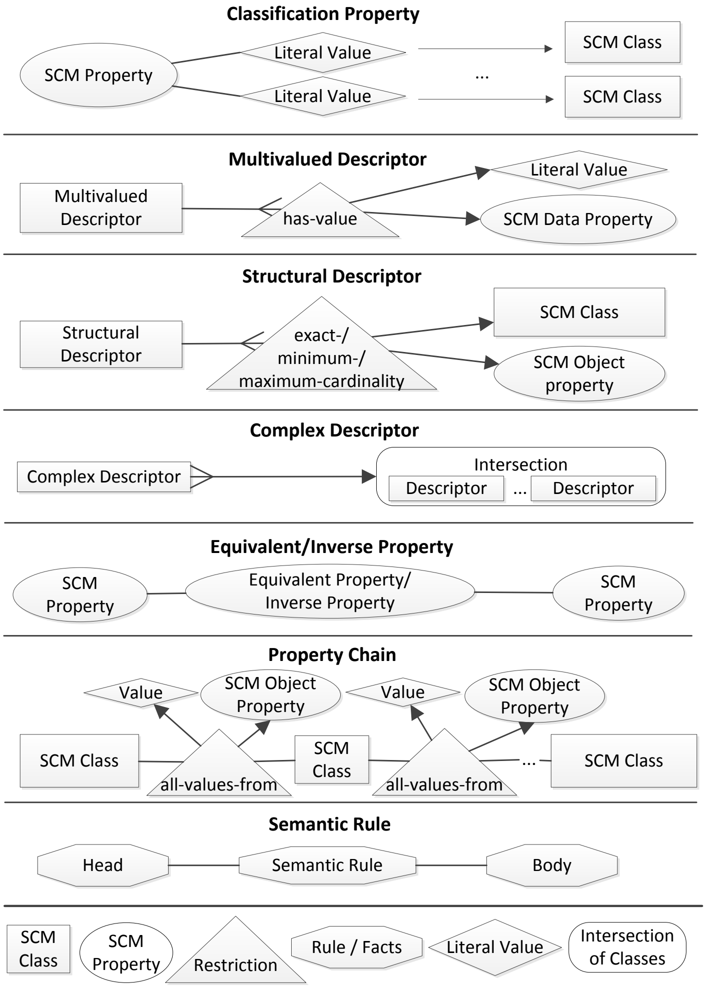


Figure 6.3: *Mapping patterns* used in *mapping guidelines*

needs to be specified as a subclass of the intersection of minimum-cardinality, exact-cardinality or maximum-cardinality restrictions. For instance, every physical object is made of a material, every animated object has an animation assigned. The linked SCM class may also be a structural descriptor, thus creating a complex structure of linked classes. In addition, structural descriptors can be extended with SCM data properties by applying the multivalued descriptor mapping pattern.

Complex Descriptor

The *complex descriptor* mapping pattern enables specification of the desirable SCM data properties and SCM object properties of SCM individuals based on multiple SCM classes that are assigned to the SCM individuals. In contrast to the previous patterns, which enable mapping *one class-to-many properties*, complex descriptors enable mapping *many classes-to-many properties*, making desirable SCM property values conditional on SCM classes assigned to the SCM individual and SCM classes not assigned to the SCM individual. For instance, the color of a wooden object is brown, while the color of a waxen object is, e.g., pink, but it also depends on the object temperature. For every distinguishable combination of SCM classes, a separate SCM class (a complex descriptor) is created and it is specified as an equivalent to the intersection of the SCM classes that are required and the complements of the SCM classes that are not required. Due to the use of the complements of SCM classes, the close world assumption has to be made to enable the use of this pattern. Every complex descriptor can be further extended with SCM data properties and SCM object properties by applying the multivalued descriptor and the structural descriptor mapping patterns.

Equivalent and Inverse Properties

The *equivalent property* mapping pattern enables specification of an SCM property as an equivalent to another SCM property. Equivalent properties are processed in the same manner. A number of SCM properties may be specified as equivalent. For instance, the `includes`, `contains` and `incorporates` SCM properties may be equivalents even if defined in different SCM ontologies. The *inverse property* mapping pattern enables the specification of an inverse SCM property [57].
Following [224], if an SCM property P_1 is an **inverse property of** an SCM property P_2, for every resources R_1 and R_2 if R_2 is related to R_1 by P_1, R_1 is related to R_2 by P_2.
$\forall P_1 \in rdf\text{:}\mathit{Property}\ \ \forall P_2 \in rdf\text{:}\mathit{Property}(\mathit{inverseOf}(P_1,P_2) \Rightarrow \forall R_1 \in rdfs\text{:}\mathit{Resource}\ \ \forall R_2 \in rdfs\text{:}\mathit{Resource}(P_1(R_1,R_2) \Rightarrow P_2(R_2,R_1)))$
For instance, `includes` and `is included in` are inverse SCM properties.

Property Chain

The *property chain* mapping pattern enables connection between SCM individuals of two different SCM classes by linking the classes via mediating SCM classes that are subclasses of all-values-from restrictions. Every mediator SCM class is specified as a subclass of an all-values-from restriction that indicates the next SCM class in the chain using an SCM object property. The linked SCM class is also a subclass of an all-values-from restriction. For instance, in an example interior design project, a room may include only objects made of wood. Indicating an object as included in the room indicates that the object is wooden.

Semantic Rule

The *semantic rule* mapping pattern is the most general of all of the CCPs proposed. It overtakes the previous patterns in terms of expressiveness. This pattern is used to create logical implications that determine selected SCM properties of SCM individuals (in the head of the rule) on the basis of other SCM properties of SCM individuals (in the body of the rule). For instance, every object `standing on` a table has the `y` coordinate calculated on the basis of the `y` coordinate of the table and the `heights` of both the objects.

6.3.3. Mapping Guidelines

The use of particular mapping patterns for creating particular mapping concepts on the basis of concrete components and concrete properties is determined by mapping guidelines as depicted in Figure 6.4. Mapping guidelines are described in the following subsections.

Creating Presentable Object Classes

For each domain-specific class C whose SCM individuals need to have independent representations in the created content, create a separate POC (defined in 5.6.3) and specify it as a superclass of C. Specify concrete data properties and concrete object properties that are inherent to the POs of the POC. Use the structural descriptor pattern to link the POC with DIC using concrete object properties (Figure 6.4-I), e.g., incorporating subobjects, indicating materials and animations. Use the multivalued descriptor pattern to assign the required concrete data properties, e.g., colors, coordinates and dimensions. In such a way,

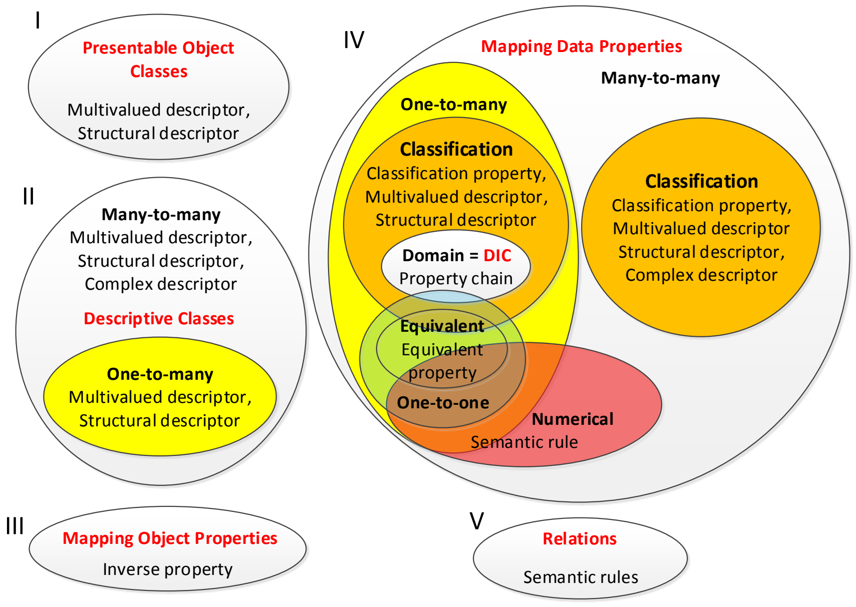

I
Presentable Object Classes
Multivalued descriptor, Structural descriptor
II
Many-to-many
Multivalued descriptor, Structural descriptor, Complex descriptor
Descriptive Classes
One-to-many
Multivalued descriptor, Structural descriptor
III
Mapping Object Properties
Inverse property
IV
Mapping Data Properties
Many-to-many
One-to-many
Classification
Classification property, Multivalued descriptor, Structural descriptor
Domain = DIC
Property chain
Equivalent
Equivalent property
One-to-one
Numerical
Semantic rule
Classification
Classification property, Multivalued descriptor Structural descriptor, Complex descriptor
V
Relations
Semantic rules

Figure 6.4: *Mapping guidelines* for creating *mapping concepts*

every domain-specific individual of C, which will be created in Step 3, will be described by all the concrete properties assigned to the POC.

If C occurs in a hierarchy of domain-specific classes, its ascendant domain-specific classes should be described first. Additional presentational effects that are not inherent to the ascendant SCM classes, should be described directly for C.

Creating Descriptive Classes

Each domain-specific class C that may be assigned to POs to specify their presentational properties but does not identify independent entities to be presented, specify as a DC and apply one of the following rules (Figure 6.4-II).

1. If C exclusively determines different concrete properties that are not collectively determined by other domain-specific classes (mapping *one class-to-many properties*), describe the C individuals by concrete object properties and concrete data properties using the structural descriptor and the multivalued descriptor patterns, respectively.
2. If C collectively determines different concrete properties with other domain-specific classes (mapping *many classes-to-many properties*), first, use the complex descriptor pattern to create a separate DC for every considered combination of the SCM classes that are assigned and the SCM classes that are not assigned to the SCM individual. Second, use the structural descriptor and the multivalued descriptor patterns for each of these DCs to specify their concrete object properties and concrete data properties.

Like in the case of mapping hierarchies of POCs, mapping hierarchies of DCs covers first mapping the ascendant DCs and second mapping the descendant DCs.

Creating Mapping Object Properties and Descriptive Individuals

For each domain-specific object property that links DIs to other DIs or DIs to POs, use the inverse property pattern to create its inverse object property, if it does not exist (Figure 6.4-III). Maintaining bidirectional links (object properties and their inverse object properties) between SCM individuals (POs and DIs) is necessary to enable application of the property chain mapping pattern (which uses SCM object properties to link DIs to DIs and DIs to POs).

Creating Mapping Data Properties

To each domain-specific data property DP that needs to have presentational effects on the described POs, apply one of the following rules (Figure 6.4-IV).

1. If DP exclusively determines particular concrete properties, regardless of other data properties, DCs and DIs assigned to the described SCM individual (mapping *one property-to-many properties*), apply one of the following rules.
   a) If the domain of DP is a POC, apply one of the following rules.
      i. If DP is equivalent to a concrete data property, indicate this fact using the equivalent property pattern (mapping *one property-to-one property*).
      ii. If DP is a classification property (its domain is a finite SCM class of literal values), use the following combination of mapping patterns. First, use the classification property pattern to create a separate DC for each distinct value of DP. Second, extend the DCs by applying structural descriptors and multivalued descriptors to assign required concrete object properties and concrete data properties to them.
   b) If the domain of DP is a DIC and its range is a set of classification data, apply the following combination of mapping patterns. First, use the classification property pattern to create a separate DC for each distinct DP value. Second, use the property chain pattern to specify the path between the DIC and the described POC. Third, extend the DCs using the structural descriptor and the multivalued descriptor to specify the required concrete object properties and concrete data properties of their POs.
2. If the range of DP is a set of numerical data, for which a formula can be specified to determine the values of the linked concrete properties on the basis of DP, use the semantic rule pattern.
3. If DP collectively determines different concrete properties in combination with other SCM data properties, DCs and DIs assigned to the POs (mapping *many properties-to-many properties*), perform the following steps. First, use the classification property pattern to specify a separate DC for every distinct value of every considered DP. Second, use the structural descriptor and the multivalued descriptor patterns to specify concrete object properties and concrete data properties of the DCs. Third, use the complex descriptor pattern to create a new DC that is the intersection of the appropriate DCs.

Like in the case of hierarchies of POCs and DCs, mapping domain-specific data properties starts with ascendant properties and only these domain-specific subproperties that introduce additional presentational effects (in comparison to their superproperties) are additionally described.

Creating Relations

Each domain-specific object property whose domain and range are POCs, specify as an RL, and create a rule for it, according to the semantic rule pattern (Figure 6.4-V), determining the values of desirable concrete properties of the participants of the RL on the basis of domain-specific properties of other participants.

Each domain-specific class C that has no independent representation in the created content, and for which there are at least two domain-specific object properties that link C with some POCs, specify as an RL and create a rule describing dependencies between particular properties of the POCs.

6.3.4. Example of Mapping Representations

In Step 2 of the considered example, a developer or a technician creates an RM (Listing 6.2—the RDF Turtle and Prolog-like syntax) including semantic statements that map domain-specific classes and properties (used in Step 3) to concrete components of the CrR (created in Step 1). The `dso:Woman` (3-4) artifact is a POC and a subclass of the `crr:WomanMesh`, so it inherits the SCM properties related to geometry that were specified in the Step 1. Every instance of `dso:Woman` can be made of clay or glass (as indicated by `dso:madeOf`), thus having an appropriate material assigned using proper DCs (6-16). In contrast to clay and glassy artifacts, every `dso:PaintedWoman` PO has a texture assigned, as indicated by its superclass (18-19). Mapping the other classes of the DSO to POCs has been performed in a similar fashion (21-22). Moreover, two basic shapes (`rm:Box` and `rm:Cylinder`) are created (24-32) and assembled into the `dso:Stand` POC (34-42). Every `rm:Box` and `rm:Cylinder` included in a `dso:Stand` has dimensions and a position (44-56). Furthermore, two RLs have been specified. The `dso:incorporates` RL is an equivalent to the `cro:includes` (58-59), while the `dso:standsOn` RL determines the `x`, `y` and `z` coordinates of an SCM individual by semantic rules (61-77). The created mapping is depicted in Figure 6.5.

Listing 6.2: A representation mapping (RM)

```
1  Prefixes:  Concrete 3D Content Ontology (cro), Concrete 3D Content Representation (crr),
       Mapping Ontology (MO), Representation Mapping (RM), Domain-specific Ontology (dso),
       Conceptual 3D Content Representation (cpr)
2
3  dso:Woman rdfs:subClassOf
4   mo:PresentableObject , crr:WomanMesh.
5
6  {rm:ClayObject,rm:GlassyObject}
7   rdf:type owl:Class ;
8   rdfs:subClassOf mo:DescriptiveClass ;
9   owl:equivalentClass
10    [ rdf:type owl:Restriction ;
11     owl:onProperty dso:madeOf ;
12     owl:hasValue "{clay,glass}" ] ,
13    [ rdf:type owl:Restriction ;
14     owl:onProperty cro:material ;
15     owl:onClass {crr:ClayMaterial,crr:GlassMaterial} ;
16     owl:qualifiedCardinality 1 ].
17
18 dso:PaintedWoman rdfs:subClassOf
19  mo:PresentableObject , crr:PaintedWomanMesh.
20
21 {dso:Granary,...,dso:Badge} rdfs:subClassOf mo:PresentableObject ,
22  {crr:GranaryMesh, ..., crr:BadgeMesh}.
23
24 {rm:Box,rm:Cylinder} rdfs:subClassOf {cro:Box,cro:Cylinder} ,
25  [ rdf:type owl:Restriction ;
26    owl:onProperty cro:size ;
27    owl:onClass {rm:BoxSize,rm:CylinderSize} ;
28    owl:qualifiedCardinality 1 ] ,
29  [ rdf:type owl:Restriction ;
30    owl:onProperty cro:position ;
31    owl:onClass {rm:BoxPos,rm:CylinderPos} ;
32    owl:qualifiedCardinality 1 ].
```

```

dso:Stand rdfs:subClassOf mo:PresentableObject , cro:StructuralComponent,
  [ rdf:type owl:Restriction ;
    owl:onProperty cro:includes ;
    owl:onClass rm:Cylinder ;
    owl:qualifiedCardinality 1 ] ,
  [ rdf:type owl:Restriction ;
    owl:onProperty cro:includes ;
    owl:onClass rm:Box ;
    owl:qualifiedCardinality 1 ].

{rm:BoxSize,rm:CylinderSize} rdf:type owl:Class ;
  rdfs:subClassOf cro:Vector ,
  [ rdf:type owl:Restriction ;
    owl:onProperty x ;
    owl:hasValue "..." ] ,
  [ rdf:type owl:Restriction ;
    owl:onProperty y ;
    owl:hasValue "..." ] ,
  [ rdf:type owl:Restriction ;
    owl:onProperty z ;
    owl:hasValue "..." ].

{rm:BoxPos,rm:CylinderPos} rdf:type owl:Class ; ...

dso:incorporates rdfs:subPropertyOf mo:BinaryRelation ;
  owl:equivalentProperty cro:includes.

dso:standsOn rdfs:subPropertyOf mo:BinaryRelation.
{cro:x(APos, AX),cro:z(APos, AZ)} :-
  dso:standsOn(A, B) ,
  cro:position(A, APos) ,
  cro:position(B, BPos) ,
  {cro:x(BPos, BX),cro:z(BPos, BZ)} ,
  {AX=BX,AZ=BZ}.
cro:y(APos, AY) :-
  dso:standsOn(A, B) ,
  cro:position(A, APos) ,
  cro:position(B, BPos) ,
  cro:y(BPos, BY) ,
  cro:size(B, BSize) ,
  cro:sy(BSize, BSY) ,
  cro:size(A, ASize) ,
  cro:sy(ASize, ASY) ,
  AY = BY + (ASY + BSY)/2.
```

6.4. Step 3: Designing a Conceptual 3D Content Representation

In this section, the concept and an example of designing CpRs in Step 3 of SCCM are discussed.

6.4.1. The Concept of Designing Conceptual Representations

Designing a CpR is a process of declarative creation of 3D content at an arbitrary level of abstraction that is permitted by the DSO used (Chapter 4/Requirements 1 and 2). This step can be performed multiple times for a particular DSO, a CrR and an RM, when new 3D content is required for a particular 3D/VR/AR application. This step focuses on the use of domain-specific classes and properties and does not cover concrete components and concrete properties, which are hidden behind the RM.

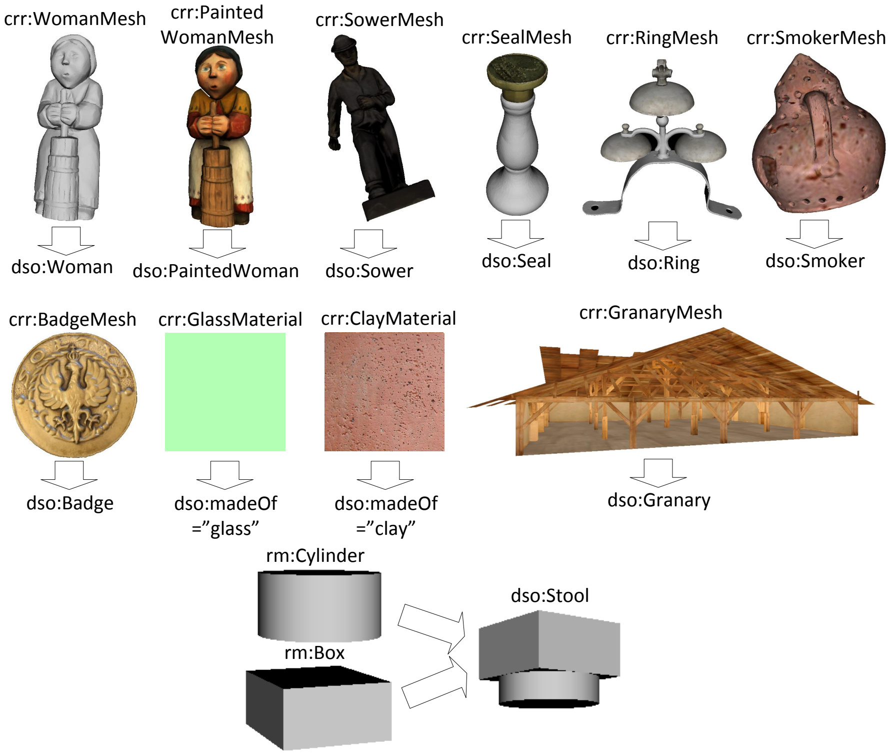


Figure 6.5: An example of mapping domain-specific concepts to concrete 3D content components

A CpR, which is an SCM knowledge base compliant with a DSO, consists of statements (interpreted as facts) and rules (interpreted as implications), which declaratively represent 3D content at the conceptual level of abstraction. Both statements and rules are on domain-specific individuals (instances of domain-specific classes) and domain-specific properties. CpRs represent coherent 3D scenes and 3D objects—in contrast to CrRs including possibly independent concrete components, which do not necessarily form coherent 3D content representations.

In general, this step is independent of the previous steps, and a CpR may be created before a CrR and an RM are created, e.g., when a domain expert designs an accurate digital equivalent to a known real object, using well-defined domain-specific concepts. However, when designing non-existing virtual 3D objects or scenes (e.g., a planned virtual museum exhibition, 3D city model) the availability of the CrR and the RM is desirable to enable preview while modeling.

This step is typically performed by a domain expert who does not need to have advanced technical skills. A domain expert uses a DSO to focus only on domain-specific semantic concepts and does not work with concrete components of 3D content.

6.4.2. Example of Designing a Conceptual Representation

In Step 3 of the considered example, a domain expert creates a CpR (Listing 6.3) including SCM individuals of domain-specific classes, which are described by domain-specific properties. Both the classes and the properties are mapped to concrete components and properties included in the CrR. The domain expert creates three woman statuettes (8-12) and several other artifacts (3-6). The first two statuettes are made of different materials (clay and glass), while the third statuette is inherently covered by a texture (as specified in the RM). Furthermore, eight stands are created (14). Their x, y and z coordinates are declaratively specified by assertions (16-24). In Line 23, the cut-off and the negation-as-failure operators are used to determine a stand, for which no x coordinate has been specified. Finally, all the artifacts and stands are incorporated in the cpr:granary (26-29).

Listing 6.3: A conceptual 3D content representation (CpR)

```
1   Prefixes:  Domain-specific Ontology (dso), Conceptual 3D Content Representation (cpr)
2
3   {dso:Granary,...,dso:Badge} rdfs:subClassOf dso:Artifact.
4
5   {cpr:granary,...,cpr:badge}
6     rdf:type {dso:Granary,...,dso:Badge}.
7
8   {cpr:clayWoman,cpr:glassyWoman}
9     rdf:type dso:Woman ;
10    dso:madeOf "{clay,glass}".
11
12  cpr:paintedWoman rdf:type dso:PaintedWoman.
13
14  {cpr:stand1,...,cpr:stand8} rdf:type dso:Stand.
15
16  cpr:standPositions(Index, N) :-
17    Index<N, dso:Stand(S), cpr:noPosition(S),
18    X is Index div 4, assert(dso:x(S, X)),
19    Z is Index mod 4, assert(dso:z(S, Z)),
20    assert(dso:y(S, 0)),
21    NewIndex is Index+1,
22    cpr:standPositions(NewIndex, N).
23  cpr:noPosition(S) :- dso:x(S, X), !, false.
24  cpr:noPosition(S).
25
26  dso:incorporates(X, Y) :-
27    dso:Granary(X),
28    (dso:Artifact(Y) ; dso:Stand(Y)),
29    X!=Y.
```

6.5. Step 4: Expanding the Conceptual 3D Content Representation

In this section, the concept, the expanding algorithm and an example of expanding CpRs in Step 4 of SCCM are discussed.

6.5.1. The Concept of Expanding Conceptual Representations

Steps 1-3 of modeling provide an SCR that represents 3D content at different levels of abstraction—concrete and conceptual. These steps cover all the activities that must be resolved by a human—the specification of how domain-specific concepts should be represented by concrete components and the specification of what domain-specific individuals should form the modeled content. So far, the concrete components of 3D content (designed in Step 1) are assigned to domain-specific concepts (classes and properties) by mapping concepts (designed in Step 2). However, the concrete components are not directly related to SCM individuals of domain-specific classes, which are described by domain-specific properties (in the CpR designed in Step 3). To enable presentation of the domain-specific individuals, the CpR is expanded according to the RM. *Expanding* is a process of creating SCM individuals of concrete components, linking them to domain-specific individuals by concrete object properties, and describing them by concrete data properties. Expanding is based on the discovery of tacit knowledge (Chapter 4/Requirement 3) and is performed in the following three stages. At Stage I, hidden facts are inferred from RLs. The facts may be related to SCM classes, properties and individuals at different levels of abstraction. At Stage II, CDs are collected for the particular POs to determine concrete object properties and concrete data properties of the POs. Finally, at Stage III, individuals of concrete properties are generated and linked to POs by concrete object properties. Also, concrete data properties are assigned to POs and to new SCM individuals generated to determine their presentational effects.

In these three stages, the CpR is expanded to a 3D meta-scene. In contrast to a CpR, a meta-scene specifies also the low-level details that are necessary for the presentation of the CpR and are imposed by the CrR. The expanding algorithm is described in the following subsection.

6.5.2. The Expanding Algorithm

The input data of the expanding algorithm is the SCR created in Steps 1-3. The SCR consists of the CrR, RM and CpR. The expanding algorithm consists of the following stages.

I. **Reasoning on RLs**: perform reasoning on RLs (created with the semantic rule mapping pattern).
Reasoning on RLs leads to the discovery of CDs, DIs and MPs assigned to POs. The discovered facts will be used at the next stages.

II. **Collecting CDs**:
   A. For all the domain-specific properties used, determine their equivalent properties (created with the equivalent property mapping pattern).
   B. For all the domain-specific object properties used, determine their inverse properties (created with the inverse property mapping pattern).
   C. For every statement on a domain-specific object property P_1 <PO_1,P_1,PO_2>, create the statement <PO_2,P_2,PO_1>, where P_2 is an inverse SCM property of P_1.
   D. To every PO, assign CDs (created with the multivalued descriptor and structural descriptor mapping patterns) on the basis of the DIs directly or indirectly linked to the PO (by all-values-from restrictions created with the property chain mapping pattern).
   E. To every PO, assign CDs (created with the classification property mapping pattern) on the basis of the MPs of the PO.
   F. To every PO, assign CDs (created with the complex descriptor mapping pattern) on the basis the CDs already assigned in Steps II-D and II-E.

As the result of completing Stage II, appropriate CDs are assigned to particular POs. The CDs describe concrete object properties and concrete data properties of the POs. Therefore, the CDs may be used for generating SCM individuals of concrete components, linking them to the POs by concrete object properties and setting concrete data properties of the POs—to provide 3D representations of the POs.

III. **Assigning properties**: for every PO, set variable I to the PO and:
   A. For every cardinality restriction with a constraint <P,R,N> that I belongs to:
      1) if there is no other cardinality restriction on P and R that has already been processed for I, create N individuals of SCM class R and link them to I by P;
      2) for every created SCM individual I_2 of R, set I to I_2 and go to Step III-A.
   B. For every has-value restriction <P,R> that I belongs to, add SCM property P with value R to I.

At this stage, inconsistencies between different restrictions should be reported, e.g., multiple orientations specified for a PO. The resulting meta-scene represents SCM individuals described by concrete data properties and related one to another by concrete object properties.

6.5.3. Example of Expanding a Conceptual Representation

In Step 4 of the considered example, the CpR is expanded according to the RM into a 3D meta-scene. The meta-scene includes eight artifacts. It is generalized, as it does not include some elements that are required for final 3D content presentation (the artifacts have no positions). Thus, only approximate presentation of the meta-scene is possible (Figure 6.6). In expanding the CpR, new SCM individuals of concrete components are generated and linked to the domain-specific individuals of the CpR by concrete object properties. Next, concrete data properties of the generated SCM individuals are properly set (Listing 6.4). All the artifacts are specified as SCM individuals of `cro:Mesh3D` (3). For the clay and the glassy woman statuettes (5-7), appropriate SCM individuals representing materials are generated (9-12). `cpr:paintedWoman` and its material are created in a similar way (14-17). Next, every `dso:Stand` is expanded to a `cro:StructuralComponent` that includes a `cro:Cylinder` and a `cro:Box` with appropriate dimensions and relative positions (21-37). For every `dso:Stand` a position is determined (39-43). Finally, all the SCM individuals (artifacts and stands) are included in the `cpr:granary` (45).

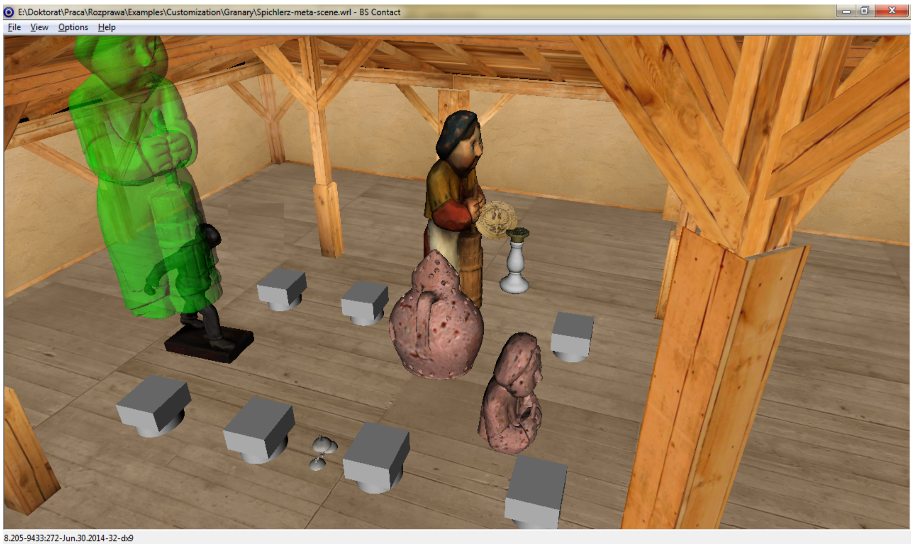

E:\Doktorat\Praca\Rozprawa\Examples\Customization\Granary\Spichlerz-meta-scene.wrl - BS Contact
File View Options Help
8.205-9433:272-Jun.30.2014-32-dx9

Figure 6.6: An approximate presentation of a 3D meta-scene

Listing 6.4: A 3D meta-scene

```
1   Prefixes:  Concrete 3D Content Ontology (cro), Conceptual 3D Content Representation (cpr),
        3D meta-scene (ms)
2
3   {cpr:granary,...,cpr:badge} rdf:type cro:Mesh3D.
4
5   {cpr:clayWoman,cpr:glassyWoman}
6     rdf:type cro:Mesh3D ;
7     cro:material {ms:clayMaterial,ms:glassMaterial}.
8
9   {ms:clayMaterial,ms:glassMaterial}
10    rdf:type {cro:TextureMaterial,cro:ColorMaterial} ;
11    {cro:texture,cro:color} {"clay.png","green"} ;
12    cro:transparency {0,0.7}.
13
14  cpr:paintedWoman rdf:type cro:Mesh3D ;
15    cro:material ms:paintedWomanMaterial.
16
17  ms:paintedWomanMaterial rdf:type cro:TextureMaterial ... .
18
```

```
{cpr:clayWoman,...,cpr:badge} cro:sensor {ms:touchSensor1,...,ms:touchSensor8}.

{cpr:stand1,...,cpr:stand8}
 rdf:type cro:StructuralComponent ;
 cro:includes {ms:cylinder1,...,ms:cylinder8} , {ms:box1,...,ms:box8}.

{ms:cylinder1,...,ms:cylinder8,ms:box1,...,ms:box8}
 rdf:type {cro:Cylinder,cro:Box} ;
 cro:size {ms:size1,...,ms:size16} ;
 cro:position {ms:pos1,...,ms:pos16}.

{ms:size1,...,ms:size16}
 rdf:type cro:Vector ;
 cro:x "..." ;
 cro:y "..." ;
 cro:z "...".

{ms:pos1,...,ms:pos16}
 rdf:type cro:Vector ; ... .

{cpr:stand1,...,cpr:stand8}
 cro:position {ms:stand1Pos,...,ms:stand8Pos}.

{ms:stand1Pos,...,ms:stand8Pos}
 rdf:type cro:Vector ;... .

cpr:granary cro:includes {cpr:clayWoman,...,cpr:badge} , {cpr:stand1,...,cpr:stand8}.
```

6.6. Step 5: Customizing the 3D Meta-Scene

In this section, the concept, the customization algorithm and an example of customizing 3D meta-scenes in Step 5 of SCCM are discussed.

6.6.1. The Concept of Customizing Meta-Scenes

3D content customization is the process of creating a semantic 3D content representation that satisfies individual requirements of a *content consumer* (Chapter 4/Requirement 5) [238]. An outline of this step is presented in Figure 6.7. A content consumer may be either a user or an application that retrieves a 3D content representation for a specific purpose. Requirements for content customization are specified in queries that conform to the *Customization Ontology* (CO).

Definition 64. *A **query** is an SCM knowledge base that:*

— *is an instance of the CO, a DSO, the CRO, a CpR and a CrR,*
— *is created according to some 3D content customization patterns.*

A query may include statements on SCM classes, properties and individuals defined in the SCM ontologies and 3D content representations.

Definition 65. *The **Customization Ontology (CO)** is an SCM ontology in which SCM classes and SCM properties used for 3D content customization are defined.*

Definition 66. *A **3D Content Customization Pattern (CCP)** is a sequence of activities performed by a content consumer that lead to solving a problem frequently occurring in 3D content customization.*

The idea of CCPs is similar to the idea of software design patterns [66]. A semantic 3D content representation that can be customized using a query (i.e., a customizable semantic 3D content representation) is referred to as a *3D meta-scene*. The general scheme of a 3D meta-scene, which complies with SCM is depicted in Figure 6.8.

Definition 67. *A **3D meta-scene** is an SCM knowledge base that is an instance the CRO, a DSO, a CrR and a CpR.*

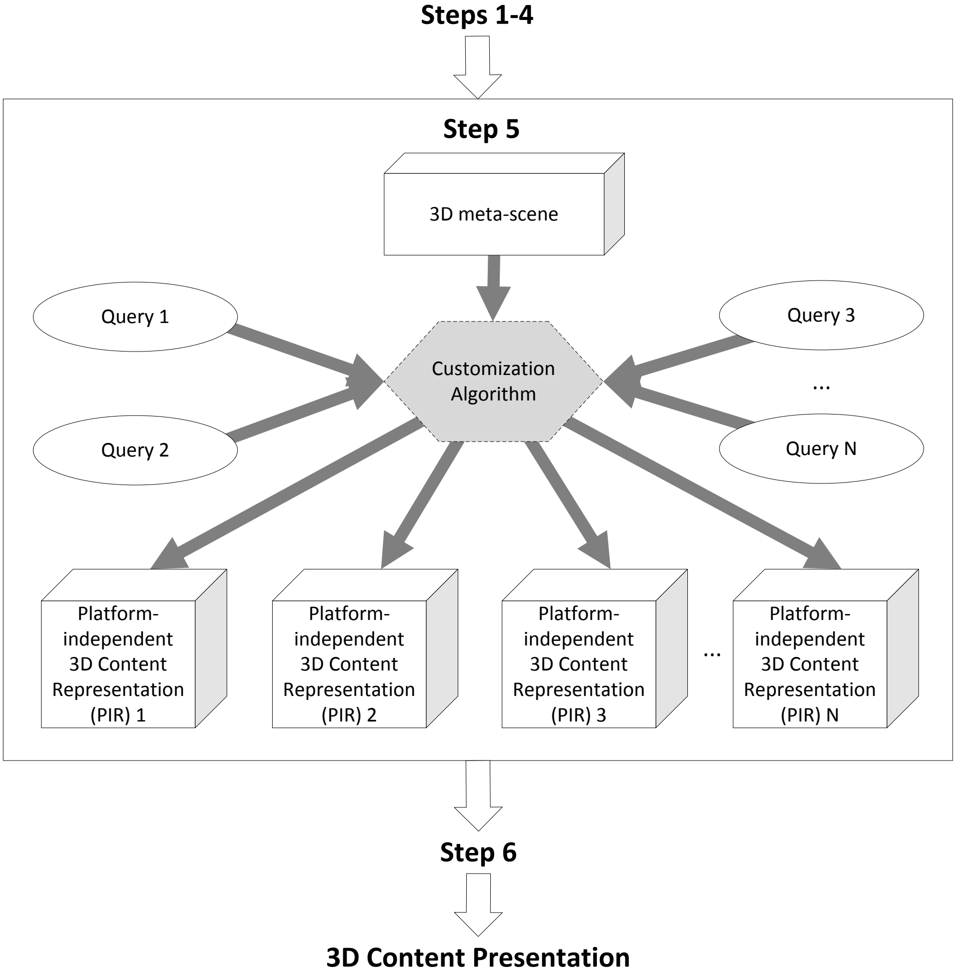

Steps 1-4
Step 5
3D meta-scene
Query 1
Query 2
Customization Algorithm
Query 3
...
Query N
Platform-independent 3D Content Representation (PIR) 1
Platform-independent 3D Content Representation (PIR) 2
Platform-independent 3D Content Representation (PIR) 3
...
Platform-independent 3D Content Representation (PIR) N
Step 6
3D Content Presentation

Figure 6.7: Step 5 of SCCM—customizing a 3D meta-scene

A 3D meta-scene is:

— *flexible*—it represents 3D content at different levels of abstraction: concrete—specific to 3D computer graphics (imposed by a CrR)—and conceptual—specific to an arbitrarily chosen application or domain (imposed by a DSO). For instance, a 3D scene of a virtual museum includes various artifacts (statuettes, armor, coins, etc.), which are combinations of different 3D objects with SCM properties;

— *generalized*—it represents a superset of presentable 3D content. For instance, a 3D meta-scene of a city includes a large number of cars each of which is described by redundant materials. Only selected cars with selected materials are presented;

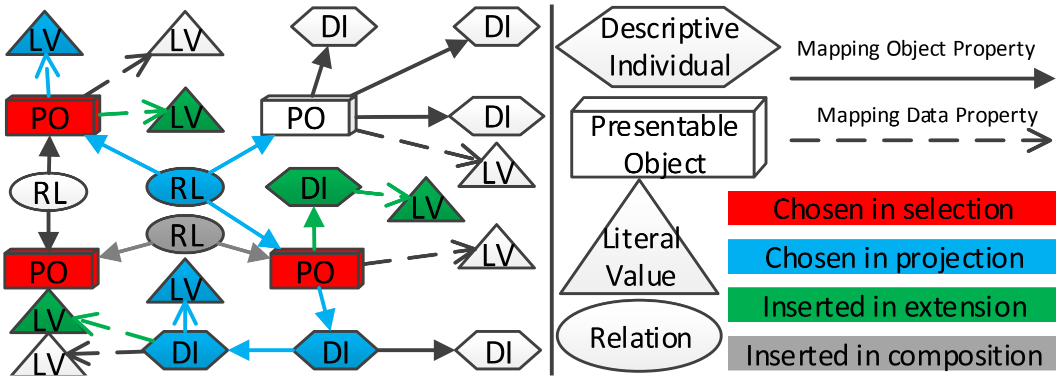


Figure 6.8: The scheme of a 3D meta-scene being customized

— *abstract*—it does not need to specify all elements that are required for final 3D content presentation. For instance, animations or positions of some cars may be unspecified in the mentioned 3D meta-scene;
— *extensible*—new elements may be added to it to create the customized 3D scene. For instance, some of the unspecified animations and positions are added within a consumer's query.

Both a 3D meta-scene and a query can be specified at different levels of abstraction by conforming to different SCM ontologies and 3D content representations. The customization of a 3D meta-scene is performed simultaneously, on demand, by different consumers independently executing individual queries to the meta-scenes. The customization results are individual PIRs (which are customized semantic 3D content representations) that satisfy the requirements of the particular consumers. Customization may cover the following four activities:

— *selection* of SCM individuals that are to be presented;
— *projection* of SCM individuals by specifying their desirable SCM properties;
— *extension* of SCM individuals by adding new SCM properties;
— *composition* of different SCM individuals by adding new relations.

Elements of a 3D meta-scene that are processed in the particular activities are depicted in Figure 6.8. The first activity (content selection) provides SCM individuals to be assembled in the generated PIR, while the other activities configure and extend the selected SCM individuals determining their final form. While content selection and content projection chose SCM individuals and SCM properties to be presented, content extension and content composition modify the SCM individuals by inserting additional SCM properties and relations.

CCPs used in content customization are depicted in Figure 6.9. In *concrete CCPs*, content consumers use semantic concepts (SCM classes and SCM properties) to build queries. *Abstract CCPs* are not directly applied to query design (are too general) but form the pattern hierarchy.

Two groups of CCPs are distinguished in terms of query processing. The first group includes CCPs in which SCM individuals and SCM properties to be presented (POs, MPs, DIs and RLs) are explicitly (directly) indicated. The processing of such CCPs is relatively simple, as it requires only basic reasoning on SCM classes, properties and individuals (classification of SCM individuals into SCM classes predefined in the CO) based on the subclass relations between SCM classes. The second group includes CCPs, in which SCM individuals are implicitly indicated—by their SCM properties and relations. The processing of such CCPs is more complicated, as in addition to the basic reasoning on

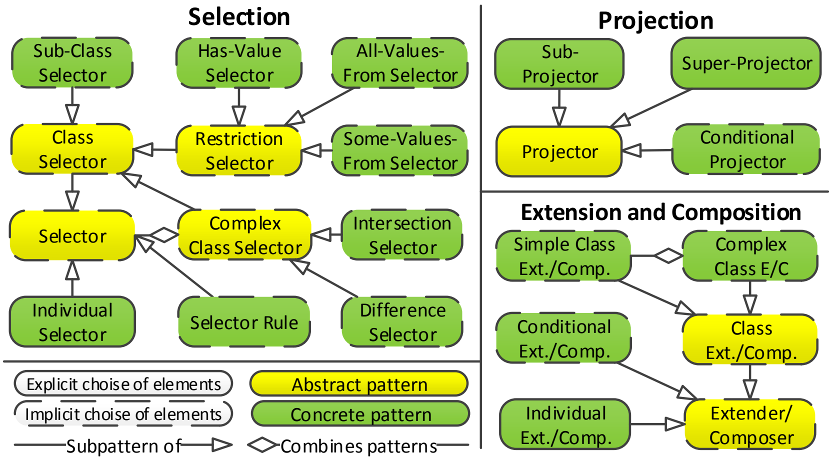


Figure 6.9: The 3D Content Customization Patterns (CCPs)

relations between classes, it also requires classification respecting SCM properties and relations of SCM individuals. Activities within particular CCPs are described in detail in Section 6.6.2.

6.6.2. 3D Content Customization Patterns

The 3D Content Customization Patterns (CCPs) proposed within SEMIC cover three content customization activities: content selection, content projection and content modification.

Content Selection

Selection of 3D content indicates desirable SCM individuals (POs, DIs and n-ary RLs) that occur in a semantic 3D meta-scene and are to be included in the customized PIR. In general, SCM individuals for a PIR may be selected explicitly or implicitly—depending on the complexity of the queried 3D meta-scene, the acceptable complexity of the query and the consumer's knowledge of the meta-scene.

Explicit Selection of Presentable Objects. In an explicit selection, SCM individuals to be presented are directly indicated in a query. SCM individuals are explicitly selected using the *individual selector* CCP in two steps. In the first step, a content consumer creates a new *SCM class* within the query and specifies it as a subclass of the `co:Selector` SCM class, which is defined in the CO. In the second step, every SCM individual that needs to be included in the generated PIR, is explicitly assigned to the created *SCM class*. The inclusion of an SCM individual in the `co:Selector` SCM class is followed by the inclusion of the SCM individual in the PIR. The individual selector CCP can be useful, in particular, when generating PIRs that incorporate single representatives of POCs, DICs or n-ary RLs that are known to the consumer (by URIs), e.g., present the models of the Empire State Building and the Statue of Liberty from a 3D meta-scene of New York.

Implicit Selection of Presentable Objects. In an implicit selection, SCM individuals to be presented are indirectly indicated in a query. SCM individuals are implicitly selected using the *class selector* CCP. This CCP can be useful when generating scenes that incorporate all the SCM individuals that have common features but are not known in advance or are not intended to be explicitly enumerated in the query (e.g., because of their large number). In this CCP, a content consumer creates a new *subclass* of the `co:Selector` SCM class. Furthermore, the new *subclass* may be used in three different ways depending on the intended purpose of the content customization.

First, the *subclass* may be specified as a superclass of a POC, DIC or n-ary RL, selecting all the SCM individuals that belong to this SCM class for the inclusion in the PIR (the *sub-class selector* CCP). This CCP is useful when all the SCM individuals of a common SCM class need to be presented (included in the generated PIR), e.g., present all the single-family houses included in a city model, present all the artifacts located in a virtual museum, etc.

Second, the *subclass* may be specified as a restriction (the *restriction selector* CCP). This CCP is useful when the presentation should include all the SCM individuals that have a common feature but do not necessarily belong to a common existing SCM class. Three restriction selector CCPs have been specified. The *has-value selector* CCP enables the presentation of SCM individuals that have a particular value of a particular SCM property, e.g., present all sculptures made of wood. The *some-values-from selector* CCP enables presentation of SCM individuals that may have multiple values of an SCM property, at least one of which belongs to a desirable SCM class, e.g., present all the artifacts that include at least one wooden element. The *all-values-from selector* CCP enables presentation of all the SCM individuals that are related to some SCM individuals by an SCM property, e.g., present artifacts that have only wooden elements.

Third, the aforementioned CCPs are based on SCM classes and SCM properties which, in general, do not enable some specific complex queries, in particular queries that condition the selection of SCM individuals from SCM properties of other SCM individuals. Semantically complex queries that combine various SCM individuals, classes and properties, are created using rules that associate SCM individuals with the created *subclass* of the `co:Selector` SCM class (the *selector rule* CCP). For instance, present all the museum artifacts that are made of the same material as the material of the David sculpture.

SCM classes created according to the previous CCPs may be combined using set operators (intersection and difference) according to the *complex class selector* CCP. This CCP is useful when SCM individuals should satisfy logically complex requirements covering multiple features (but not semantically complex requirements as in the case of the selector rule CCP), e.g., present all the single-family houses that are made of wood and have double-glazed windows (the *intersection selector* CCP), but their roofs are not made of asbestos (the *difference selector* CCP).

Content Projection

Projection of 3D content indicates desirable features of SCM individuals (chosen in content selection) that are to be included in the PIR. The features of SCM individuals are encoded by MDPs with literals, MOPs with DIs as well as RLs.

Content projection is performed in a few steps using the `projector` CCP, which permits the choice of MPs and binary RLs. In the first step, a content consumer creates a new *SCM property* (an SCM data property or an SCM object property) and specifies it as a subproperty/subrelation of the `co:Projector` property defined in the CO. All the subproperties/subrelations of `co:Projector` that are specified for some SCM individuals, are included in the generated PIR. Hence, all the MPs/RLs that are the subproperties of the created *property/binary RL* become presentable. The second step of the CCP depends on the complexity of the queried meta-scene and the acceptable complexity of the created query.

First, the created *SCM property/binary RL* may be specified as a superconcept of the chosen MP/binary RL (the *sub-projector* CCP). In this way, the MP will be included in the generated PIR, as it becomes a subproperty of `co:Projector`. This CCP is convenient for presenting a particular

MP/binary RL as well as all MPs/binary RLs with a common root concept for all the SCM individuals included in the generated PIR. For example, the CCP can be used to present all SCM properties related to appearance (color, transparency, shininess, etc.) for all POs in the scene, present all RLs that determine relative locations of POs, etc.

Second, the created *SCM property/binary RL* may be specified as a subproperty of a desirable MDP, MOP or a binary RL and—in addition—as a subproperty of `co:Projector` (the *super-projector* CCP). Next, for a statement in the meta-scene that needs to be included in the PIR, a new *statement* is added to the query. The new *statement* includes the same subject and the same object as the base statement, but exchanges the base MP/binary RL with the created *SCM property/binary RL*. The created *SCM property/binary RL* inherits all presentational effects from its base concept, so it may be used in the same way in modeling content. However, the explicit choice of the created *SCM property/binary RL* instead of its base concept, enables the contextual presentation of the base concept (only for the designated SCM individual) and prevents the presentation of the base concept for the other SCM individuals. For instance, present all knights in a game but only the king should be clad in armour.

In addition to the aforementioned CCPs for the explicit choice of MPs and binary RLs, which is unconditional—determined in advance in the query, it is also possible to implicitly choose MPs/binary RLs depending on specific conditions. In the *conditional projector* CCP, a MP/binary RL is classified as a subproperty of the created *SCM property/binary RL* by a rule, e.g., present colors of POs only if all the POs in the meta-scene have colors (for preserving the consistency of the generated PIR).

Projection of n-ary RLs, which are encoded using DIs related to SCM individuals by MOPs, is performed using CCPs designed for selection of SCM individuals (cf. Section 6.6.2/*Content Selection*).

Content Modification

Content modification encompasses *content extension* and *content composition*. Both activities introduce new elements that are responsible for new features or behavior of the selected SCM individuals. Content extension introduces new MPs describing selected SCM individuals, while content composition introduces new RLs combining selected SCM individuals by specifying dependencies expressed by MPs. Since the encoding of queries and the encoding of meta-scenes are uniform—based on the semantic web standards—content modification may be performed in the same way as in the case of creating SCRs.

Neither an extension nor a composition may change the MPs and RLs that are already set to some values. Otherwise, the specification of such MP/RL leads to an inconsistency in the generated PIR. To enable the modification of an MP/RL that already has a value in the meta-scene, a new generated PIR in which the SCM property/RL is filtered out, needs to be created by a query to the meta-scene. Furthermore, the generated PIR should be used as a new meta-scene for a new query, which may extend the new meta-scene with a new value of the desirable MP/RL (content customization chain).

Like content selection, content modification—including extension and composition—may be accomplished explicitly or implicitly, using the *extender/composer* CCPs.

Explicit Content Modification. In explicit content modification, new MPs are directly assigned to selected SCM individuals (the *individual extender/composer* CCP). For instance, present a particular building from a geometrical model of a housing estate and assign a color to it; select a model of the Earth and a model of the Moon and add an RL moving the Moon around the Earth.

Implicit Content Modification. In implicit content modification, new MPs are assigned to a POC, DIC or n-ary RL (the *class extender/composer* CCP) using restrictions (like in the case of the restriction selector CCP). If a MDP is modified, the *simple class extender/composer* CCP is used to create a new *has-value restriction* that indicates a desirable value of the MDP. Next, the SCM class is specified as an equivalent to a previously created *class selector* SCM class. In this way, all the SCM individuals that belong to the *class selector* SCM class in query processing, also have the desirable MP indicated by the created *has-value restriction*. For instance, select all wooden objects and add shininess to their materials.

If an MOP is modified, the *simple class extender/composer* CCP is used to create a new *cardinality restriction* that indicates the desirable range the MP. Next, the SCM class is specified as an equivalent to a previously created *class selector* SCM class (like in the case of modifying an MDP). In content customization, a new SCM individual of the SCM class that is in the MOP range is created and connected to the primary SCM individual by the MOP. For instance, select all POs that have no material specified and assign wooden material to each of them.

Specification of multiple new SCM individuals to be introduced requires the use of the *complex class extender/composer* CCP, which is created using the intersection of the SCM classes created according to the simple extender/composer CCP—in a similar manner as in the complex class selector CCP.

A conditional modification of an SCM individual that depends on MPs of other SCM individuals, may be performed similarly to a conditional projection of an SCM individual (using a rule—the *conditional extender/composer*), e.g., set color of the ceiling to a value that is equal to the color of the walls in the room.

6.6.3. The Customization Algorithm

The input data of the customization algorithm is the union of the meta-scene (generated in Step 4) and a query. The customization algorithm has three stages related to different activities permitted by the CCPs: content selection, content modification and content projection. Content selection and content modification interlace. On the one hand, the results of a selection are used in succeeding modification, e.g., add transparency (modification) to all objects that are covered by a texture (selection). On the other hand, the results of a modification may affect the selection and projection, e.g., once red color is set to an SCM individual, the SCM individual is selected for presentation as one of the colorful SCM individuals in a meta-scene. Some tasks within the stages, which are mutually independent, are completed in parallel, e.g., selecting SCM individuals or SCM properties using different selector/projector CCPs. To simplify the description of the customization algorithm, SCM classes, SCM properties and rules created according to CCPs are referred to using the names of the CCPs, e.g., an SCM class created with the *sub-class selector CCP* is referred to as a *sub-class selector*, an SCM property created with the *sub-projector* CCP is referred to as a *sub-projector* and so on. The exact description of the stages of the customization algorithm is presented below.

I. **Selecting content**:
   A. Create a PIR that is a copy of the input meta-scene.
      The PIR will be further gradually reduced by removing statements that are neither on selected SCM individuals nor on selected SCM properties.
   B. Infer which SCM individuals are instances of `co:Selector` (as implicitly specified) by running in parallel:
      1) every SCM individual of an *individual selector* set as an instance of `co:Selector`;
      2) every SCM individual of a subclass of a *sub-class selector* set as an instance of `co:Selector`;

3) every SCM individual that satisfies the has-value constraint of a *has-value selector* set as an instance of `co:Selector`;
4) every SCM individual that satisfies the all-values-from constraint of an *all-values-from selector* set as an instance of `co:Selector`;
5) every SCM individual that satisfies the some-values-from constraint of a *some-values-from selector* set as an instance of `co:Selector`;
6) every SCM individual that belongs to an *intersection selector* set as an instance of `co:Selector`;
7) every SCM individual that belongs to a *difference selector* set as an instance of `co:Selector`;
8) infer which SCM individuals belong to `co:Selector`, using *selector rules*.

   The steps directly assign SCM individuals to the `co:Selector` *class. Only such SCM individuals will be included in the customized PIR.*

II. **Modifying content**:

A. Run the expanding algorithm to generate new SCM individuals on the basis of *extender/composer* classes that are equivalents of *selector* classes, and link the generated SCM individuals to the appropriate SCM individuals that are already included in the meta-scene (cf. Section 6.5).

   Running the expanding algorithm is necessary to complement the structure and the SCM properties that are imposed by the modification.

B. If any SCM individual has been modified, go to Step I-B to select SCM individuals whose SCM properties changed in the content modification in Step II-A, and which satisfy requirements given in the query.

   As the result of the modification, some new SCM individuals may satisfy requirements for presentation. Such SCM individuals should be verified again at Stage I.

C. For every SCM individual I that does not belong to `co:Selector`, remove all the statements on I (in which I is either the subject or the object) from the PIR.

III. **Projecting content**:

A. Infer which SCM properties are subproperties of `co:Projector` (as implicitly specified) by running in parallel:

   1) every subproperty of a *sub-projector* set as a subproperty of `co:Projector`;
   2) infer which SCM properties belong to `co:Projector` by using the *conditional projector* CCP.

B. For every SCM property P that does not belong to `co:Projector`, remove all the statements on P from the PIR.

6.6.4. Example of Customizing a Meta-Scene

In Step 5 of the considered example, the meta-scene is customized according to requirements specified in three queries: A, B and C. The queries were prepared according to CCPs (Listing 6.5). All the queries select `cpr:granary` and all the SCM properties defined in the DSO for presentation (Lines 3-4).

Query A uses the selector rule CCP to declaratively assign one artifact to one stand (7-11). Every artifact that is not on any stand, is placed on a stand on which there is no artifact yet. It is only important to deploy all the artifacts on some stands, but it is not important, on which stand a particular artifact is placed. In the rules, negation as failure and cut-off are used. An artifact is placed on a stand by calculating the X, Y and Z coordinates of the artifact with respect to the coordinates of the stand (13-16). The resulting PIR is presented in Figure 6.10.

Query B extends Query A with the selection of only the artifacts that are made of clay and their stands. The selection of the artifacts is achieved using the sub-class selector and some-values-from selector CCPs (19-25). The selection of the stands is achieved by creating the inverse SCM property of `dso:standsOn` (27-28) as well as the sub-class selector and restriction selector CCPs (30-36). The resulting PIR is presented in Figure 6.11.

Query C extends Query B by using the selector rule CCP to chose only the artifacts between which the distance is maximal (39-44). The rule verifying the maximal distance has been implemented using negation as failure and cut-off (like in Lines 8,10), and it is skipped in the listing. Furthermore, the simple class extender CCP is used. For every `q:Object`, concrete components responsible for rotating the object after it is touched are created (according to the restrictions) and assigned (46-50)—`q:TouchSensor` (52-57), which is activated by `q:RotatingInterpolator` (59-70), which is controlled by `q:TimeSensor` (72-76). The resulting PIR is presented in Figure 6.12.

Listing 6.5: Queries to a 3D meta-scene

```
1  Prefixes:  Concrete 3D Content Ontology (cro), Domain-specific Ontology (dso), Conceptual 3D
       Content Representation (cpr), Customization Ontology (co), Query (q)
2
3  cpr:granary rdf:type co:Selector.
4  dso:Property rdfs:subPropertyOf co:Projector.
5
6  # --------- Query A ---------
7  q:standsOn(A, B) :- dso:Artifact(A), dso:Stand(B), q:notOnOthers(A), q:nothingOnIt(B).
8  q:notOnOthers(A) :- q:standsOn(A, B), !, fail().
9  q:notOnOthers(A).
10 q:nothingOnIt(B) :- q:standsOn(A, B), !, fail().
11 q:nothingOnIt(B).
12
13 cro:x(A, AX) :- q:standsOn(A, B), cro:x(B, BX), AX = BX.
14 cro:z(A, AZ) :- q:standsOn(A, B), cro:z(B, BZ), AZ = BZ.
15 cro:y(A, AY) :- q:standsOn(A, B), cro:y(B, BY), cro:height(B, BHeight), cro:height(A, AHeight),
16   AY = BY+(AHeight+BHeight)/2.
17
18 # --------- Query B ---------
19 q:ClayObject
20   rdf:type owl:Class ;
21   rdfs:subClassOf co:Selector.
22   owl:equivalentClass
23     [ rdf:type owl:Restriction ;
24       owl:onProperty dso:madeOf ;
25       owl:hasValue "clay" ] ;
26
27 q:inverseStandsOn rdf:type rdf:Property ;
28   owl:inverseOf dso:standsOn.
29
30 q:SelectedStand
31   rdf:type owl:Class ;
32   rdfs:subClassOf co:Selector.
33   owl:equivalentClass
34     [ rdf:type owl:Restriction ;
35       owl:onProperty q:inverseStandsOn ;
36       owl:someValuesFrom q:ClayObject ] ;
37
38 # --------- Query C ---------
39 q:remote(A, B) :-
40   cro:x(A, AX), cro:x(B, BX),
41   cro:z(A, ZX), cro:z(B, ZX),
 Distance^2 = (AX-BX)^2 + (AZ-BZ)^2,
 maximal(Distance^2).
co:Selector(A) :- q:remote(A, B).

q:Object rdfs:subClassOf
 [ rdf:type owl:Restriction ;
   owl:onProperty cro:sensor ;
   owl:onClass q:TouchSensor ;
   owl:qualifiedCardinality 1 ].

q:TouchSensor rdf:type owl:Class ;
 rdfs:subClassOf cro:TouchSensor ,
 [ rdf:type owl:Restriction ;
   owl:onProperty cro:activates ;
   owl:onClass q:RotatingInterpolator ;
   owl:qualifiedCardinality 1 ].

q:RotatingInterpolator rdf:type owl:Class ;
 rdfs:subClassOf cro:OrientationInterpolator ,
 [ rdf:type owl:Restriction ;
   owl:onProperty cro:key ;
   owl:hasValue "0 1.5 3" ] ,
 [ rdf:type owl:Restriction ;
   owl:onProperty cro:keyValue ;
   owl:hasValue"0 0 0 0 3.14 0 0 0 0"],
 [ rdf:type owl:Restriction ;
   owl:onProperty cro:controller ;
   owl:onClass q:TimeSensor ;
   owl:qualifiedCardinality 1 ].

q:TimeSensor rdf:type owl:Class ;
 rdfs:subClassOf cro:TimeSensor,
 [ rdf:type owl:Restriction ;
   owl:onProperty cro:interval ;
   owl:hasValue 3 ].
```

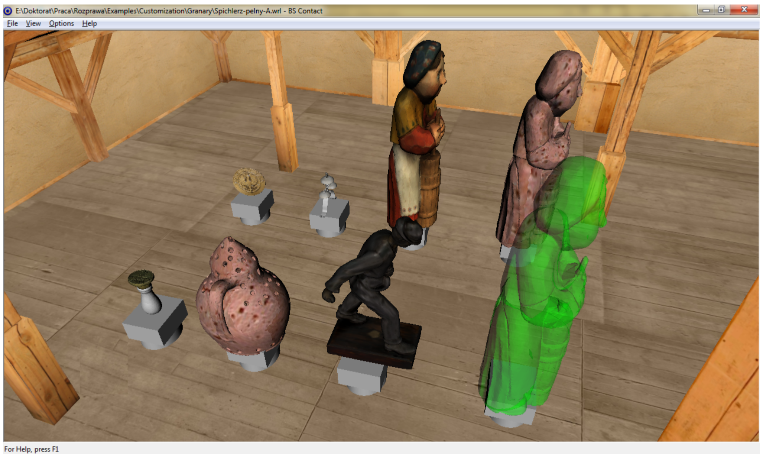

E:\Doktorat\Praca\Rozprawa\Examples\Customization\Granary\Spichlerz-pelny-A.wrl - BS Contact
File View Options Help
For Help, press F1

Figure 6.10: An example of a customized PIR, in which every artifact is assigned to one stand

6.7. Step 6: Generating Platform-specific 3D Content Representations

In this section, the concept, the transformation algorithm and an example of generating PSRs upon PIRs in Step 6 of SCCM are discussed.

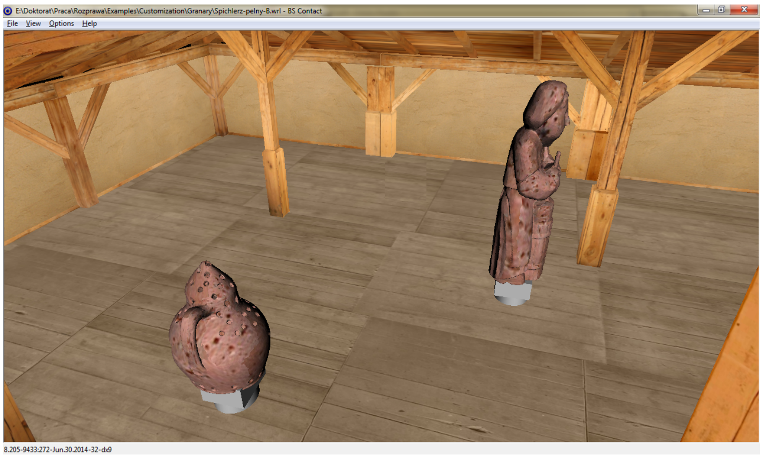

E:\Doktorat\Praca\Rozprawa\Examples\Customization\Granary\Spichlerz-pelny-B.wrl - BS Contact
File View Options Help
8.205-9433:272-Jun.30.2014-32-dx9

Figure 6.11: An example of a customized PIR including only the artifacts made of clay and their stands

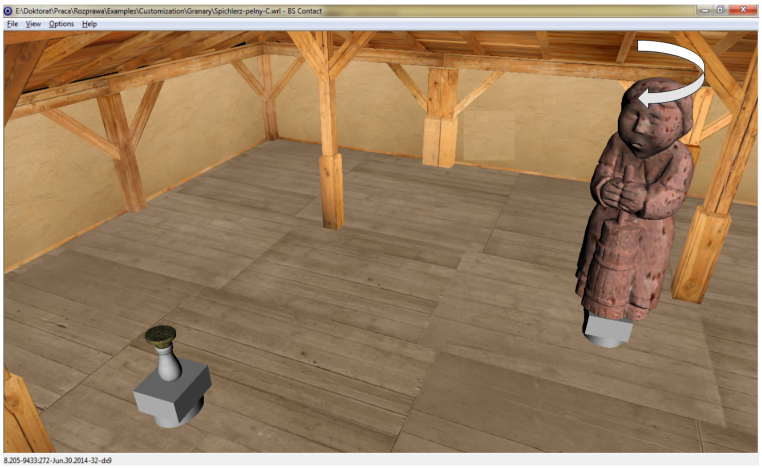

E:\Doktorat\Praca\Rozprawa\Examples\Customization\Granary\Spichlerz-pelny-C.wrl - BS Contact
File View Options Help
8.205-9433:272-Jun.30.2014-32-dx9

Figure 6.12: An example of a customized PIR including only the objects between which the distance is maximal, and which rotate after being touched

6.7.1. The Concept of Generating Platform-specific Representations

Step 6 of SCCM enables generation of 3D content for a variety of content presentation platforms (Chapter 4/Requirement 6) [132, 140, 141]. An outline of this step is presented in Figure 6.13. PIRs are SCM knowledge bases that represent 3D content at both concrete and conceptual levels of abstraction, using the CRO and a DSO, respectively. The transformation algorithm processes PIRs at the concrete level of abstraction and generates PSRs using TKBs and TBs. PSRs are documents encoded in arbitrarily selected 3D content representation languages, thus PSRs may be presented using various platforms—3D content browsers and presentation tools installed on different devices. A pair of a TKB and a TB is designed specifically for a particular content presentation platform that implements a particular language. A TKB describes the rules of transformation. A TB is a set of parametrized templates encoded in a 3D content representation language. During the transformation, templates are filled with parameters and combined into PSRs according to the TKB. In SCCM, once a PIR is created, it may be automatically transformed to several PSRs.

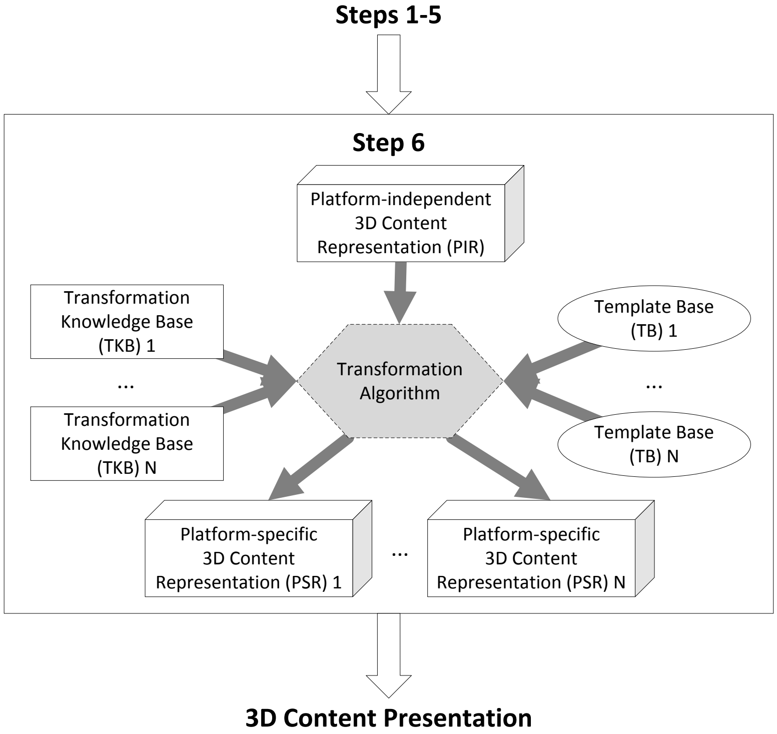

Steps 1-5
Step 6
Platform-independent 3D Content Representation (PIR)
Transformation Knowledge Base (TKB) 1
...
Transformation Knowledge Base (TKB) N
Transformation Algorithm
Template Base (TB) 1
...
Template Base (TB) N
Platform-specific 3D Content Representation (PSR) 1
...
Platform-specific 3D Content Representation (PSR) N
3D Content Presentation

Figure 6.13: Step 6 of SCCM—generating platform-specific 3D content representations

6.7.2. The Transformation Algorithm

The input data of the transformation algorithm is the PIR generated in Step 5, a TKB and a TB. In the description of the transformation algorithm, the abbreviations introduced in Section 5.7 are used. The general idea of the transformation algorithm is based on the following elementary operations on Ts:

1) *setting common values of TPs* that are associated with different Ss included in common SSs,
2) *nesting Ts into other Ts* that are associated with different Ss included in common SSs,
3) *ordering Ts* that are associated with different Ss included in common SLs.

The operations are performed regarding a description of transformation that is included in the appropriate TKB (associated with the selected 3D content representation language). Due to the use of basic operations on code templates, the transformation algorithm is generic and it can generate 3D content representations encoded in different languages—both imperative and declarative. The transformation algorithm has five stages, each of which comprises several steps. First, semantic queries are created on the basis of SSPts specified in the TKB. Second, the queries are issued against a PIR to create SSs. Third, the values of TPs associated with the Ss included in the SSs are set. Next, Ts associated with the Ss are nested one into another. Finally, Ts are enumerated in the required order. The exact description of the stages of the transformation algorithm is presented below.

I. **Creating Queries**
At this stage, queries to PIRs are created on the basis of a TKB that is appropriate for the selected 3D content representation language. The queries are encoded in a semantic query language, e.g., SPARQL. Since SPARQL is the most common language for querying knowledge bases, in this description, the queries are referred to as SPARQL queries. Each SPARQL query created at this

stage corresponds to an individual SSPt. The clauses of a SPARQL query (triples `[subject, property, object])` *correspond to particular SPts that are included in the SSPt.*

For every SSPt from the TKB, create a SPARQL query as follows:

A. Create an empty SPARQL query.

B. For every SPt from the SSPt, which is given as `[?subject ?property ?object.]`:

   1) append the following clause to the SPARQL query
      `[?prop rdfs:subPropertyOf ?property.]`;

   2) if the `object` of the SPt is a literal value or an individual, append the following clause to the SPARQL query `[?subject ?prop ?object.]`;

   3) if the `object` of the SPt is an SCM class, append the following clauses to the SPARQL query `[?subject ?prop ?obj. ?obj rdfs:subClassOf ?object.]`.

Inserting the `rdfs:subPropertyOf` *and the* `rdfs:subClassOf` *properties to the SPARQL query permits not only to search for Ss of a PIR that exactly match the SPt (its* `SCM property` *and its* `object`*), but also to search for Ss that use subproperties of the* `SCM property` *and subclasses of the* `object` *that are in the SPt.*

II. **Creating Statement Sets**

At this stage, the SPARQL queries created at Stage I are issued against a PIR to create SSs. Each resulting SS is a group of logically related Ss.

For every SPARQL query created at Stage I:

A. Issue the SPARQL query against the PIR and create an SS incorporating all the Ss from the PIR that are included in the result of the SPARQL query.

B. For every S, remember the SPt from the SPARQL query that has been matched to the S.

The SSs created at this stage will be further used for setting common TPs for different Ts and for nesting some Ts into other Ts.

III. **Setting Template Parameters**

At this stage, TPs of logically related Ts are fixed regarding dependencies between the Ss the Ts are associated with.

For every SS created at Stage II, for every S, which is given as `[?subject ?property ?object]` and is included in the SS:

A. Load a TS whose signature (the concatenation of the URI of the SCM class the `subject` belongs to, the URI of the `SCM property` and the URI of the SCM class the `object` belongs to) matches the S.

B. Query the TKB for TPPts that are included in the TPts that are included in the TSPt linked to the SPt that is associated with the S:

```
[?TPPt to:isParameterOf ?TPt.  ?TPt to:isIn ?TSPt.
?TSPt to:isTemplateSetOf ?SPt.].
```

   Remember the determined *template parameter pattern set* (TPPtS) associated with the S.

C. For every TPPt$_1$ that is included in the TPPtS and has not yet been set:

   1) set a unique value of TPPt$_1$;

   2) if TPPt$_1$ is a literal parameter of the S, replace the TP that is reflected by TPPt$_1$ and is included in a T with the literal value of the S (the S object);

   3) query the TKB for TPPts that are equal to TPPt$_1$ and are linked to SPts that occur in a common SSPt with the SPt that is associated with the S:

```
[?TPPt1 to:equalTo ?TPPt2.  ?TPPt2 to:isParameterOf ?TPt.
```

`?TPt to:isIn ?TSPt.  ?TSPt to:isTemplateSetOf ?SPt`$_2$`.`
`?SPt`$_2$` to:isIn ?SSPt.  ?SPt to:isIn ?SSPt.];`

4) for every TPPt$_2$ found, set the value of the reflected TP (which is included in a T) to the value of TPPt$_1$ and recursively go to Step III-C-3. A T whose all TPs are set is a CF.

As the result of this stage, Ts have TPs set to proper literal values and common identifiers of variables. Recursive processing of TPs ensures assigning a common value to all equal TPs across different Ts.

IV. **Nesting Templates**

At this stage, Ts are nested one into another to create a hierarchical structure. Nesting is indicated by assigning Ts to TPs.

For every SS created at Stage II, for every S that is included in the SS and has not yet been processed, for every TPPt from the TPPtS (determined in Step III-B) that is associated with the SPt associated with the S:

A. Query the TKB for TPts that are equal to the TPPt and are assigned to SPts included in a common SSPt together with the SPt of the S:
`[?TPPt to:equalTo ?TPt.  ?TPt to:isIn ?TSPt.`
`?TSPt to:isTemplateSetOf ?SPt`$_2$`.  ?SPt`$_2$` to:isIn ?SSPt.  ?SPt`
`to:isIn ?SSPt.].`

B. For every TPt found, go recursively to Step IV-A.

C. Replace all the TPs that are reflected by the TPPts with the appropriate Ts that are reflected by the TPts—to produce CFs.

As setting TPs, nesting Ts is performed recursively—processing a T continues until all Ts that are to be nested in the T are processed (its appropriate TPs are set to appropriate Ts).

V. **Ordering Templates**

At this stage, Ts linked with mutually dependent Ss that need to occur in the generated PSR at the same level (without nesting one T into another) are set in the PSR in a suitable order. First, the global list of SPts is created. Second, the Ss of the PIR are sorted with respect to the global list.

A. Create the *global SLPt* list. Perform the step until all SPts that are included in different SLPts are included in the *global SLPt*:

1) if the SPt is not yet included in the *global SLPt*, but it is included in any SLPt and there are no other SPts that precede the SPt in at least one SLPt, add the SPt to the end of the *global SLPt*.

B. Add the remaining SPts (the SPts that are not included in any SLPts) to the end of the *global SLPt* in an arbitrary order.

C. Browse the *global SLPt* in the reverse order and for every SPt included in the list:

1) find Ss in the PIR that match the SPt and add their CFs (generated at the Stages I-IV) at the beginning of the generated PSR.

6.7.3. Example of Generating Platform-specific Representations

In Step 6 of the considered example, a PSR is generated on the basis of the PIR that was created using Query C in Step 5 (Figure 6.12, Listing 6.6—the X3D/XML syntax). For both the artifacts (`cpr:clayWoman` and `cpr:seal`), `Transform` nodes with positions indicated by `translation` attributes are generated (e.g., 8-28). The `Transform` nodes include `Shape` nodes with `Material` and `Texture` nodes. Moreover, the `cpr:clayWoman` artifact is equipped with a `TouchSensor` node, an `OrientationInterpolator` node and a `TimeSensor` node, which are connected by ROUTE nodes. These nodes enable rotation of the artifact after touching it. Stools are generated as Transform nodes including two shapes represented by Cylinder and Box nodes with positions and scales (e.g., 38-51). All nodes are enclosed within the common Transform node (6-54).

Listing 6.6: A generated platform-specific (X3D) content representation

```
 1 <?xml version="1.0" encoding="UTF-8"?>
 2 <!DOCTYPE ...">
 3 <X3D ... >
 4 <head>...</head>
 5 <Scene>
 6   <Transform>
 7     <Group>
 8       <Transform DEF="clayWoman" translation="...">
 9         <Shape>
10           <Appearance>
11             <Material transparency="0" />
12             <ImageTexture url="clay.png" />
13           </Appearance>
14           <IndexedFaceSet coordIndex="...">
15             <Coordinate point="..."/>
16           </IndexedFaceSet>
17         </Shape>
18         <TouchSensor DEF="touchSensor1" enabled="false"/>
19         <OrientationInterpolator DEF="rotatingInterp1" key="0 1.5 3"
20             keyValue="0 0 0 0 3.14 0 0 0 0" />
21         <TimeSensor DEF="timeSensor1" cycleInterval="3"/>
22         <ROUTE fromNode=’touchSensor1’ fromField=’touchTime’
23             toNode=’timeSensor1’ toField=’startTime’/>
24         <ROUTE fromNode=’timeSensor1’ fromField=’fraction_changed’
25             toNode=’rotatingInterp1’ toField=’set_fraction’/>
26         <ROUTE fromNode=’rotatingInterp1’ fromField=’value_changed’
27             toNode=’clayWoman’ toField=’rotation’/>
28       </Transform>
29 
30       <Transform DEF="seal" translation="...">
31         <Shape>seal model ...</Shape>
32       </Transform>
33 
34       <Transform DEF="granary" translation="...">
35         <Shape>granary model ...</Shape>
36       </Transform>...
37 
38       <Transform DEF="stool1" translation="...">
39         <Transform DEF="box1" translation="..." scale="...">
40           <Shape>
41             <Appearance><Material /></Appearance>
42             <Box />
43           </Shape>
44         </Transform>
45         <Transform DEF="cylinder1" translation="..." scale="...">
46           <Shape>
47             <Appearance><Material /></Appearance>
48             <Cylinder />
49           </Shape>
50         </Transform>
51       </Transform>
52       <!-- stool2, ..., stool8 -->
53     </Group>
54   </Transform>
55 </Scene>
56 </X3D>
```

Generating the PSR is based on a TB and a TKB whose excerpts are presented in Listings 6.7 and Listing 6.8. In the example, assigning the texture to `cpr:clayWoman` is explained. The TB includes two templates: a template determining a shape (Listing 6.7, Lines 2-7) and a template determining appearance (10-14). The templates may be combined by the use of the `elements` parameters. The combination of the templates is described in the TKB. The TKB includes an SSPt (Listing 6.8, Lines 3-4) incorporating three SPts. `tkb:definitionSPt` (6-23) is used to match Ss that define meshes, e.g., `[cpr:clayWoman rdf:type cro:Mesh3D.]`. `tkb:texturingSPt` (25-39) is used to match Ss that indicate textures of materials, e.g., `[ms:clayMaterial cro:texture "clay.png".]`. Since `tkb:pathTPP` reflects a literal value, its corresponding TP is directly exchanged with the matching value in the T. Both the SPts are connected by `tkb:intermediarySPt` (41-44). `tkb:intermediarySPt` indicates that a defined `subject` is linked to an `object` by the `cro:material` property. The SPARQL query generated for the SSPt, which matches the Ss in the PIR is presented in Listing 6.9. After matching the Ss to the SPts, the Ts are matched to the Ss using the signatures given in Listing 6.7 (Lines 1, 9). The remaining Ss of the PIR are processed in a similar way, using Ts and SPts that have been omitted in the listings.

Listing 6.7: An excerpt of a template base for X3D

```
1  Template <cro:Component-rdf:type-rdf:Class>:
2  <Transform &attributes>
3     &elements
4     <Shape>
5        ...
6     </Shape>
7  </Transform>
8
9  Template <cro:TextureMaterial-cro:TexturePath-rdfs:Literal>:
10 <Appearance>
11    &elements
12    <Material &attributes/>
13    <ImageTexture url="&path" />
14 </Appearance>
```

Listing 6.8: An excerpt of a transformation knowledge base for X3D

```
1  Prefixes:  Transformation Ontology (TO), Transformation Knowledge Base (TKB)
2
3  tkb:texturingSSPt a to:SSPt ;
4     to:hasSPt tkb:definitionSPt , tkb:texturingSPt , tkb:intermediarySPt ;
5
6  tkb:definitionSPt a to:SPt ;
7     rdf:subject "subject" ;
8     rdf:predicate rdf:type ;
9     rdf:object cro:Mesh3D ;
10    to:hasTemplateSetPattern tkb:tsp1 .
11
12 tkb:tsp1 a to:TemplateSetPattern ;
13    to:hasTemplatePattern tkb:tp1 .
14
15 tkb:tp1 a to:TemplatePattern ;
16    to:hasTemplateParameterPattern tkb:par1 , tkb:par2 .
17
18 tkb:par1 a to:TemplateParameterPattern ;
19    to:name "attributes" .
20
21 tkb:par2 a to:TemplateParameterPattern ;
22    to:name "elements" ;
23    to:equalTo tkb:texturingSPt .
```

```

tkb:texturingSPt a to:SPt ;
   rdf:subject "object" ;
   rdf:predicate cro:texture ;
   rdf:object rdfs:Literal ;
   to:hasTemplateSetPattern tkb:tsp2 .

tkb:tsp2 a to:TemplateSetPattern ;
   to:hasTemplatePattern tkb:tp2 .

tkb:tp2 a to:TemplatePattern ;
   to:hasTemplateParameterPattern tkb:pathTPP .

tkb:pathTPP a to:TemplateParameterPattern;
   to:name "path" ;
   to:isLiteral "true" .

tkb:intermediarySPt a to:SPt ;
   rdf:subject "subject" ;
   rdf:predicate cro:material ;
   rdf:object "object" .
```

Listing 6.9: A SPARQL query generated upon the TKB

```
SELECT DISTINCT * WHERE {
   ?subject ?prop1 ?obj1.
   ?prop1 rdfs:subPropertyOf* rdf:type.
   ?obj1 rdfs:subClassOf* cro:Mesh3D.

   ?subject cro:material ?object.

   ?object ?prop2 ?value.
   ?prop2 rdfs:subPropertyOf* cro:texture. }
```

7. The SO-SEMIC Environment

The *Service-Oriented Environment for Semantic Modeling of Interactive 3D Content* (SO-SEMIC) has been developed on the basis of the SEMIC approach and service-oriented architecture (SOA) [142]. *SOA is a technique that involves interaction between loosely coupled services that function independently*. [67]. SOA permits to implement flexible systems of modeling content, which enable division of responsibilities between different services and clients distributed across the web. In SOA, services may encompass complex, computationally intensive semantic processing on servers, while clients (users' applications) may perform graphical operations on users' devices. Also, SOA facilitates collaboration of users equipped with different devices on 3D content creation.

In this chapter, the implemented prototype environment SO-SEMIC is described. First, the overall architecture of the environment is explained. Second, the particular modules of the environment, which correspond to the subsequent steps of SCCM, are presented. Finally, libraries for generating PSRs are described.

7.1. System Architecture

The architecture of SO-SEMIC is depicted in Figure 7.1. The environment comprises a client (*Client*) and a server (*Server*) that implement a RESTful SOA. The Client is a Python-based application that enables presentation and visual manipulation of 3D content in the Blender modeling tool [25], as well as transformation of Blender PSRs to its equivalent PIRs. The Server is a Java application that enables manipulation of SCRs (CrRs and CpRs) using the SCM ontologies (CRO, MO and DSO). Both applications have multi-layered architectures, which are discussed in detail in the following sections.

7.1.1. Client

The multi-layered architecture of the Client conforms to the Model-View-Controller (MVC) [53] design pattern. It consists of four layers: the *Semantic Logic Layer* (controller), the *Presentation Layer* (view) as well as the *Network Layer* and the *Data Layer* (model). The Client has been implemented using the Blender API [26]. Blender has been selected for the SO-SEMIC implementation because it is an advanced, widely-used, open-source environment with extensive documentation, tutorials and several versions available for different operating systems. However, the Client could be developed using other 3D content modeling environments.

The Presentation Layer is responsible for handling requests from a user and presenting 3D content in Blender. It comprises new Blender GUI elements (panels) as well as the Blender *Graphics Engine*. The Graphics Engine allows a user to present and manipulate 3D content in the graphical environment. It enables access to a number of Blender tools, such as transformations, editors and scene graphs. The Graphics Engine is accessible through specific classes and properties of the Blender API, e.g., `bpy.data` and `bpy.context.` The new GUI elements include: the *Concrete Component Panel*, the *Concrete Property Panel*, the *Class Mapping Panel*, the *Property Mapping Panel*, the *Conceptual Design*

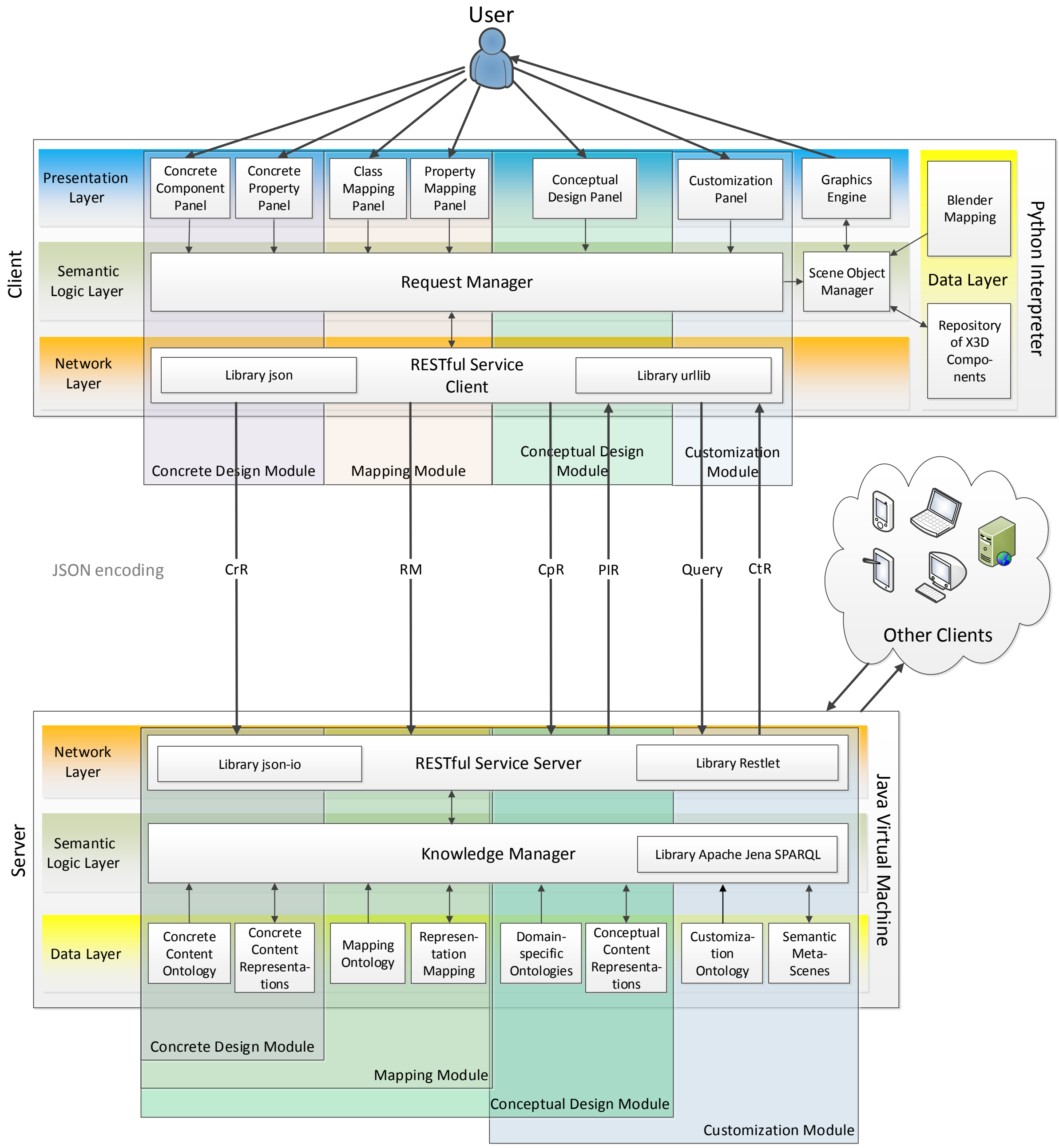


Figure 7.1: Architecture of the SO-SEMIC environment. The arrows indicate the flow of information

Panel and the *Customization Panel*. The panels, which extend the standard Blender GUI are instances of the `bpy.types.Panel` class, and they include buttons (instances of the `bpy.types.Operator` class), menus (instances of the `bpy.types.Menu` class) as well as properties (instances of the `bpy.props.StringProperty` and `bpy.props.BoolProperty` classes). Whereas the panels included in the Presentation Layer are strictly related to the particular modules of the Client, the components of the other layers are shared by different modules. The functions of particular panels are discussed in detail in Sections 7.2-7.5, which are devoted to the modules of SO-SEMIC.

The Semantic Logic Layer is responsible for processing user requests as well as creating and managing Blender PSRs that are to be presented to a user or converted into SCRs and sent to the Server. User's requests prepared with panels of the Presentation Layer (e.g., create a new domain-specific object) are received by the *Request Manager*, which invokes appropriate methods of the *Scene Object Manager* and uses the Network Layer to communicate with the Server (e.g., get the list of domain-specific classes). The Scene Object Manager uses the Graphics Engine to create, modify or remove objects in the Blender-specific scene. Further, the Scene Object Manager uses the *Data Layer* to perform transformations between Blender-specific objects and scenes and their semantic counterparts. Therefore, the Scene Object Manager is capable of processing the CRO, which is a common semantic notation of 3D content components and properties that is understandable to different clients connected to the Server.

The Data Layer is responsible for providing basic components for creating Blender PSRs. The layer specifies links between Blender PSRs (including instances of Blender-specific components, e.g., `bpy.data.lamps`, `bpy.data.cameras`) and CrRs (including concrete components and concrete properties, e.g., `cro:LightSource`, `cro:Camera`), which are readable and processable to the Server. The links enable bi-directional transformation of both types of content representations. A Blender PSR is transformed to its semantic equivalent (a PIR), e.g., when an object has been modified in Blender and it should be updated in the corresponding CpR. The transformation of a PIR is performed every time it is retrieved from the Server and should be loaded into Blender. The links are specified in the *Blender Mapping*, an excerpt of which is presented in Listing 7.1.

Listing 7.1: An example of mapping Blender-specific properties to concrete semantic properties

```
#Blender-specific properties (the left side), concrete semantic properties (the right side)

#Geometrical Properties
angle = angle

#Presentational Properties
active_material = hasMaterial
active_texture = hasTexture
filepath = uri
image = _empty_image
diffuse_color = hasDiffuseColor
r = r
g = g
b = b
alpha = transparency

#Spatial Properties
scale = hasSize
location = hasPosition
rotation_euler = hasOrientation
x = x
y = y
z = z

#Structural Properties
parent = isIncludedIn

#Light Properties
energy = intensity
```

Repository of X3D Components is a collection of X3D documents that are encoded Blender-specific equivalents of concrete components (included in a CrR). X3D has been selected to be used in SO-SEMIC, because it is a widely-used standard with an extensive documentation and a number of examples. *X3D Components* are retrieved from the *Repository* and combined by the Scene Object Manager into complex Blender PSRs. For instance, a texture, which is an X3D Component and an equivalent to a `cro:Texture` concrete component, is applied to a material, which is another X3D Component and an equivalent to a `cro:Material` concrete component. X3D Components are partially independent of their equivalent concrete components, since the properties that are not semantically specified in a concrete component may be arbitrarily set for its equivalent X3D Component. For instance, at a high level of abstraction, semantic content creation does not necessarily need to require access to the coordinates of particular vertices of a mesh, which may be specified only in X3D Components. Hence, different clients may have independent repositories of X3D Components that have different values of properties that are unspecified for concrete components, e.g., Blender PSRs of a tree may differ in two clients in terms of size, shape, color of leafs, etc.

The Network Layer is responsible for communication with the Server. The *RESTful Client Service* leverages two Python libraries. The *json library* [52] is used to encode and decode PIRs in JSON. The *urllib library* [68] is used to create, send and receive HTTP requests (which incorporate encoded PIRs), e.g., semantic queries for on-demand content generation. As a response to a request, the *RESTful Client Service* typically gets an acknowledgment or a PIR, which is further transformed and presented in Blender.

7.1.2. Server

The Server manages and provides PIRs to clients. The Server is independent of particular 3D content representation standards, content browsers and modeling environments. Hence, it could be used with different software clients installed on various devices. The multi-layered architecture of the Server consists of three layers: the *Network Layer*, the *Semantic Logic Layer* and the *Data Layer*.

The Network Layer is responsible for communication with clients. The *RESTful Server Service* leverages two Java libraries. The *Restlet* library [63] is used to receive HTTP requests and to create and send HTTP responses to clients. For every URI that may be the target of a client request, an individual Java method has been implemented. A method is invoked once a request is handled for the URI. Every method is implemented as a function of the *Semantic Logic Layer*. For instance, sending a request to the URI `/CpR1` may result in obtaining the whole CpR with all its domain-specific objects. The *json-io* library is used to decode SCRs included in the incoming requests and to encode SCRs that are sent to clients.

The Semantic Logic Layer is responsible for processing client requests as well as creating and managing PIRs, which will be stored in the *Data Layer* or will be provided to a client. The *Knowledge Manager* is based on the Apache Jena SPARQL library [78], and it retrieves, performs reasoning on and modifies SCM ontologies and knowledge bases stored in the Data Layer. The implemented inference is based on the SPARQL `CONSTRUCT` clause, which has been used instead of libraries strictly designed for OWL-based reasoning. Such approach allowed to achieve better efficiency through precise inferring only the required facts and avoiding the entailments that are not relevant to a particular case. For instance, determining super-classes of a class in the CRO is based only on the analysis of the `rdfs:subClassOf` property (explicitly specified) and does not require determining super-properties of the SCM properties included in the SCM ontology.

The Data Layer is responsible for storing SCRs and PIRs as well as the SCM ontologies to which the representations conform. Within the layer, the particular modules use different SCM ontologies that are parts of SCM, and operate on different SCM knowledge bases that are parts of SCRs representing the content at different levels of abstraction (as discussed in Chapters 5-6). Multiple content representations within a module may be assigned to different clients or users modeling 3D content. The SCM ontologies are only read for processing SCRs and PIRs, and are not modified, while SCRs and PIRs are read and modified by the Knowledge Manager. Since the CRO specifies the basic elements of 3D objects and scenes that are sent to clients, the clients must interpret this SCM ontology to transform SCRs and PSRs to representations that are understandable to the clients (e.g., Blender PSRs).

Since the Client uses RESTful web services implemented in the Server, the names of HTTP methods and fragments of URIs are used in the dissertation to identify appropriate web services, e.g., `GET /<ontology>/classes/<class>/properties` retrieves all SCM properties of the `class` specified in the `ontology`—both of which are parameters of the URI. Any HTTP request sent by the Client to the Server is initiated by the *Request Manager* and consecutively encoded in JSON, transmitted through the *RESTful Service Client*, handled by the *RESTful Service Server* and decoded. On the server-side, user requests are processed by the Knowledge Manager. Any transformation between a Blender PSR and an SCR or a PIR involves the Scene Object Manager, which uses the Blender Mapping and X3D Components. Presentation of 3D content on the client-side is performed by the Graphics Engine.

In the following sections, the particular modules of the environment are presented. The modules enable the subsequent steps of SEMIC that are accomplished by modeling users. A typical view of the Blender environment with the installed SO-SEMIC client is depicted in Figure 7.2. The SO-SEMIC panels extend the `Tools` space in the `3D View` region in Blender. The presented view is sufficient for performing the main actions of modeling 3D content, including all actions permitted by the SO-SEMIC environment (implemented panels—Figure 7.2/1) as well as setting the most common properties of 3D content elements using the tools that are inherent to Blender (the panels `Outliner`—Figure 7.2/2 and `Preferences`—Figure 7.2/3). The results of modeling are visible in the `User Perspective` window (Figure 7.2/4).

7.2. The Concrete Design Module

The *Concrete Design Module* enables accomplishment of Step 1 of SCCM—designing a CrR (cf. Section 6.2). The GUI of the module consists of two panels (Figure 7.3)—the *Concrete Component Panel* (Figure 7.3/1) and the *Concrete Property Panel* (Figure 7.3/2), which allow a user to perform actions on CrRs. The panels communicate with the *Request Manager*, which uses the *Scene Object Manager* to manage Blender-specific components and properties and produce their equivalent CrRs. CrRs are delivered to the Server and further forwarded to the Knowledge Manager, which adds them to the CrRs that are already stored on the Server. The Concrete Design Module permits the following actions.

7.2.1. Selecting a Repository of X3D Components

Selection of a Repository of X3D Components, which includes Blender-specific counterparts to concrete components, and to which new Blender-specific components may be added, is a necessary step of designing a CrR (Figure 7.4/a/1). In the implemented prototype, the Repository is a local file system directory, however, in general it could be a remote database. The selection of a Repository is performed

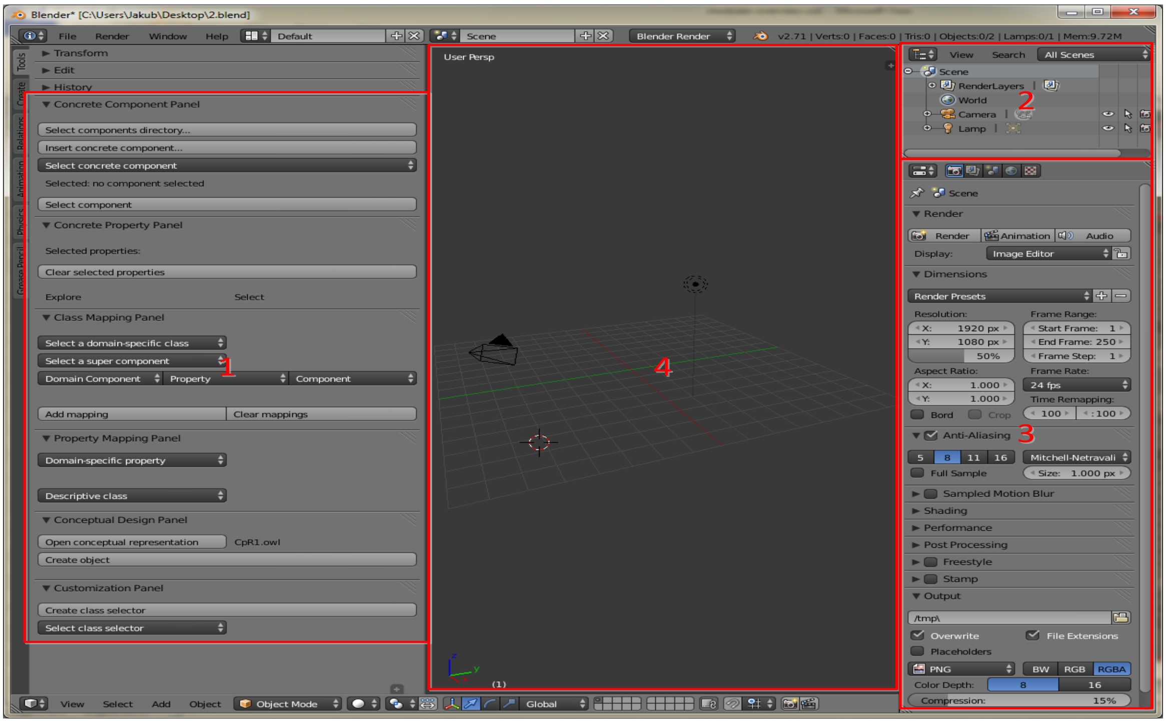


Figure 7.2: The main window of the Blender modeling tool with the installed SO-SEMIC client (1)

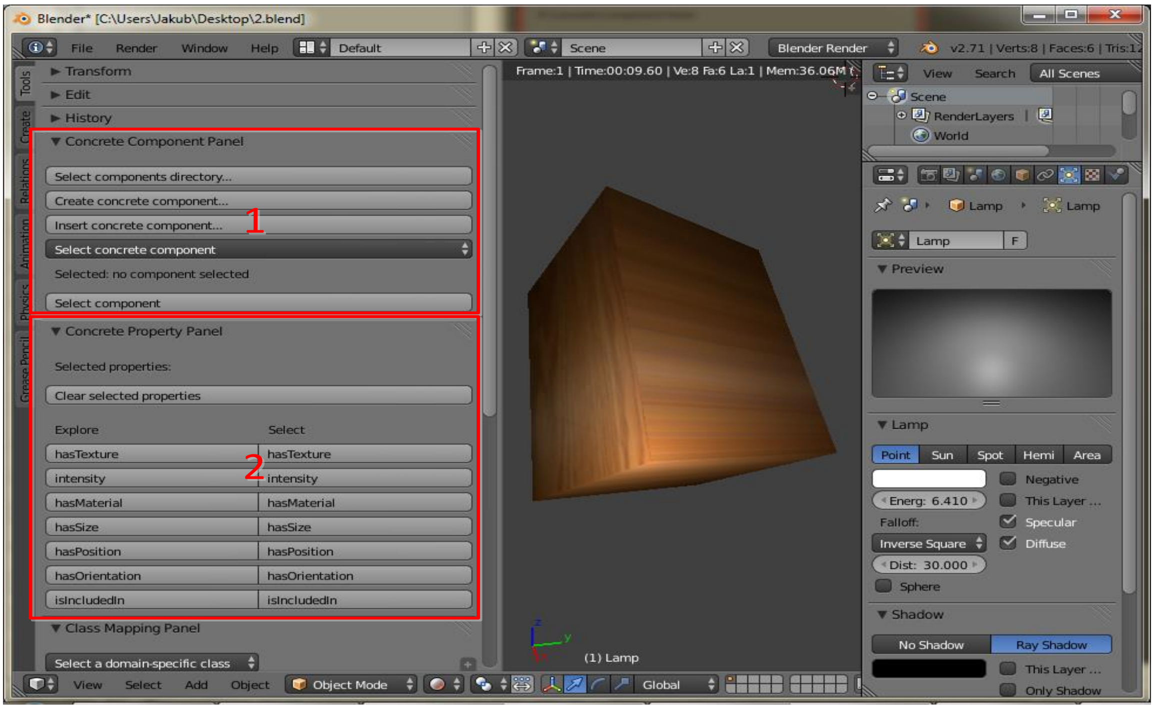

Blender* [C:\Users\Jakub\Desktop\2.blend]
Concrete Component Panel
Select components directory...
Create concrete component...
Insert concrete component...
Select concrete component
Selected: no component selected
Select component
Concrete Property Panel
Selected properties:
Clear selected properties
Explore
Select
hasTexture
intensity
hasMaterial
hasSize
hasPosition
hasOrientation
isIncludedIn
Class Mapping Panel
Select a domain-specific class
1
2

Figure 7.3: The panels of the *Concrete Design Module*

in the Client irrespectively of the Server. The X3D Components of the Repository should correspond to concrete components of the CrR used, which is stored on the Server.

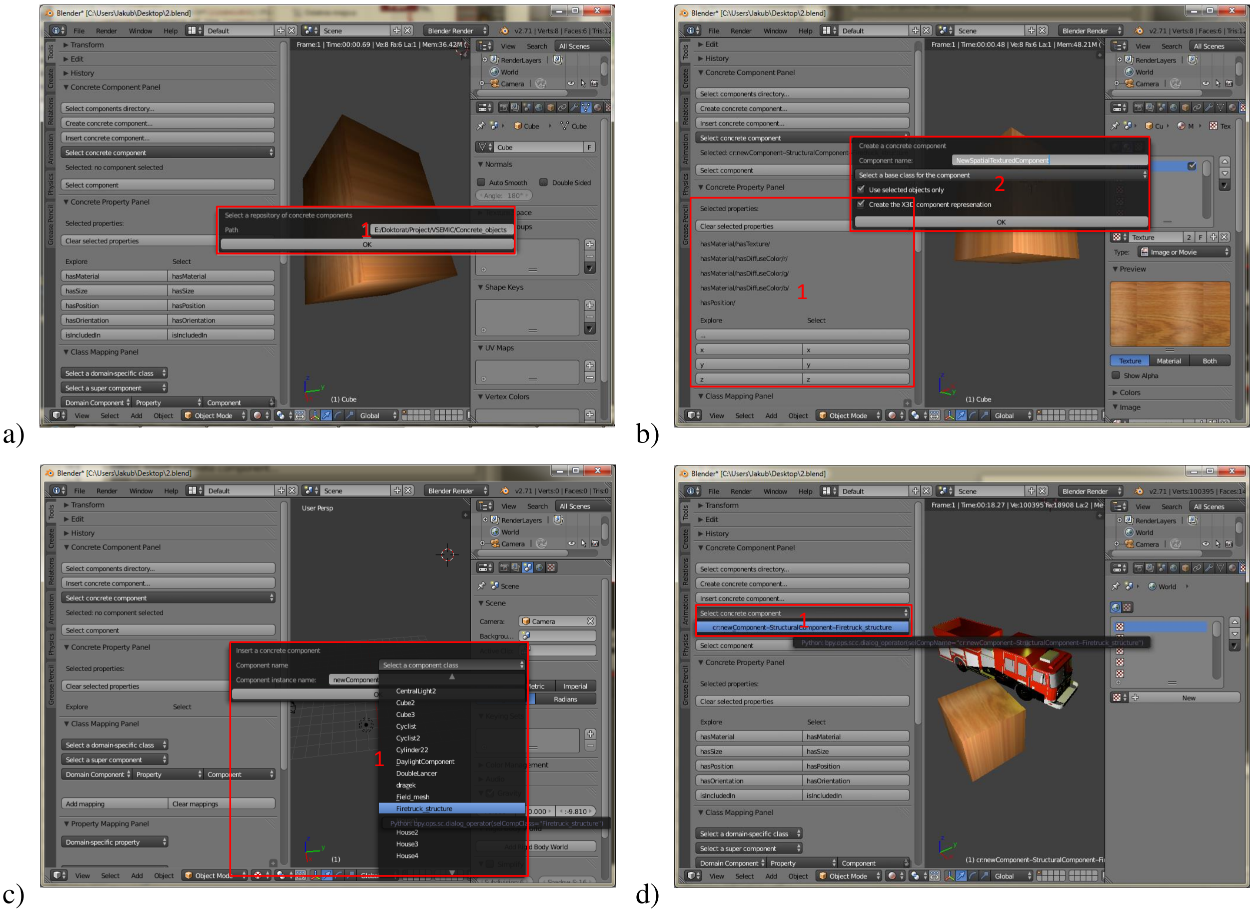

a)
Concrete Component Panel
Select components directory...
Create concrete component...
Insert concrete component...
Select concrete component
Select a repository of concrete components
Path
E:/Doktorat/Project/VSEMIC/Concrete_objects
OK
1
b)
Create a concrete component
Component name:
NewSpatialTexturedComponent
Select a base class for the component
Use selected objects only
Create the X3D component represenation
OK
1
2
c)
Insert a concrete component
Component name
Component instance name:
Select a component class
CentralLight2
Cube2
Cube3
Cyclist
Cyclist2
Cylinder22
DaylightComponent
DoubleLancer
drazek
Field_mesh
Firetruck_structure
House2
House3
House4
1
d)
Select concrete component
cr:newComponent-StructuralComponent-Firetruck_structure
1

Figure 7.4: Actions available in the *Concrete Design Module*: selection of a *Repository of X3D Components* (a), creation of a new concrete component (b), insertion of a new instance of a concrete component into a scene (c) and selection of an instance of a concrete component (d)

7.2.2. Creating Concrete Components

Creation of a concrete component requires the selection of a Blender-specific object in the scene that (or whose component) will be the prototype of the new concrete component. Depending on the Blender-specific object selected, the actual list of the available SCM object properties is presented in the Concrete Property Panel (Figure 7.4/b/1). The list is determined by the Scene Object Manager, which analyses the currently selected Blender-specific object and links its Blender-specific properties to concrete properties using the Blender Mapping. For instance, the Blender-specific property `energy` of a `Lamp` object is mapped to the concrete property `cro:intensity` of the `cro:LightSource` class. A user selects the desirable concrete properties in the list to be included in the new concrete component. For instance, an `cro:AppearanceComponent` may be described by a `cro:Material` including three color components (RGB). It is possible to create a component based on the entire scene or only on the selected Blender-specific object (Figure 7.4/b/2).

After receiving the list of concrete properties selected, the Request Manager uses the service `GET /CRO/classes` to retrieve the list of classes defined in the CRO. On the Server side, the classes are obtained by the Knowledge Manager from the CRO. The user chooses one of the classes as the base class of the new concrete component, e.g., `cro:Mesh3D`, `cro:StructuralComponent`, `cro:Texture`, etc. If a user selects a class (e.g., `cro:TextureMaterial`) that does not correspond to the selected Blender-specific object (e.g., `cro:Mesh3D`), the service `GET`

/CRO/classes/<selectedClass>/properties is invoked to get the list of the concrete properties that are applicable to the selected class, against which the selected concrete properties are verified. Only the successfully verified selected concrete properties are included in the concrete component to preserve its consistency with the CRO.

Next, an X3D Component can be created and stored in the Repository. The creation of the X3D Component facilitates further modifications of its equivalent concrete component, and determines the properties of the concrete component that are not reflected semantically. Such an approach excludes data that do not need to be semantically reflected (e.g., may be large in general—vertices of a mesh, sub-objects of a complex structural object) from concrete components and enhances semantic processing performed by the Knowledge Manager. If a user does not create the X3D Component, the use of the concrete component in a scene only implies presentational effects that are determined by the semantic statements of the concrete component. An X3D Component always encompasses the whole Blender-specific object selected, even if only its sub-component has been used to create a concrete component. For instance, a cube included in an X3D Component provides a means of presenting a texture that has been selected to create a concrete component.

The Scene Object Manager retrieves the selected concrete properties and assigns the values to them from the Blender-specific properties accessed through the Graphics Engine. Upon receiving the concrete component, the Request Manager uses the service PUT /<CrR>/individuals/<concreteComponent>. The Knowledge Manager adds the received new concrete component to the CrR specified in the requested URI. Since a concrete component is an OWL restriction class, the Knowledge Manager transforms the received CrR describing an SCM individual with SCM properties into a CrR describing a class with restrictions on SCM properties. For instance, the statement crr:compOrient cro:x 3.14 is transformed into the OWL restriction class crr:CompOrientClass rdf:type owl:Restriction. crr:CompOrientClass owl:onProperty cro:x. crr:CompOrientClass owl:hasValue 3.14. (cf. Chapter 5).

7.2.3. Inserting Concrete Components into a Scene

The insertion of an instance of a concrete component into a scene requires a user to select the name of the desirable concrete component and the name of the component instance that will be inserted into the scene (Figure 7.4/c/1). The list of the concrete components specified in the CrR is obtained using the service GET /<CrR>/classes and presented to the user. The Scene Object Manager loads from the Repository the appropriate X3D Component (if exists) that is the counterpart to the inserted concrete component, and injects it into the scene. Further, the properties of the concrete component are set to its equivalent in the scene.

7.2.4. Selecting Concrete Components

The selection of an instance of a concrete component is accomplished using a combobox that includes the identifiers of all instances of concrete components that have already been inserted into the scene (Figure 7.4/d/1). The list is created by the Scene Object Manager, which analyses the scene using the Graphics Engine. The selection performed by the Graphics Engine starts with the currently selected objects and searches for an ascendant object (parent) that is a Blender-specific counterpart to the concrete component. The selection of a concrete component instance facilitates its manipulation in Blender and the creation of new concrete components derived from the selected concrete component.

7.3. The Mapping Module

The *Mapping Module* enables accomplishment of Step 2 of SCCM—mapping domain-specific concepts to a CrR (cf. Section 6.3). The GUI of the module consists of two panels—the *Class Mapping Panel* (Figure 7.5/1) and the *Property Mapping Panel* (Figure 7.5/2). Concrete components and concrete properties linked to domain-specific concepts determine 3D representation of the domain-specific concepts, which are abstract in the sense of presentation. A user specifies desirable mappings using the panels. The mappings are delivered to the Knowledge Manager, which operates on the MO, CrRs and RMs, to which new mappings are added. The Mapping Module permits the following actions.

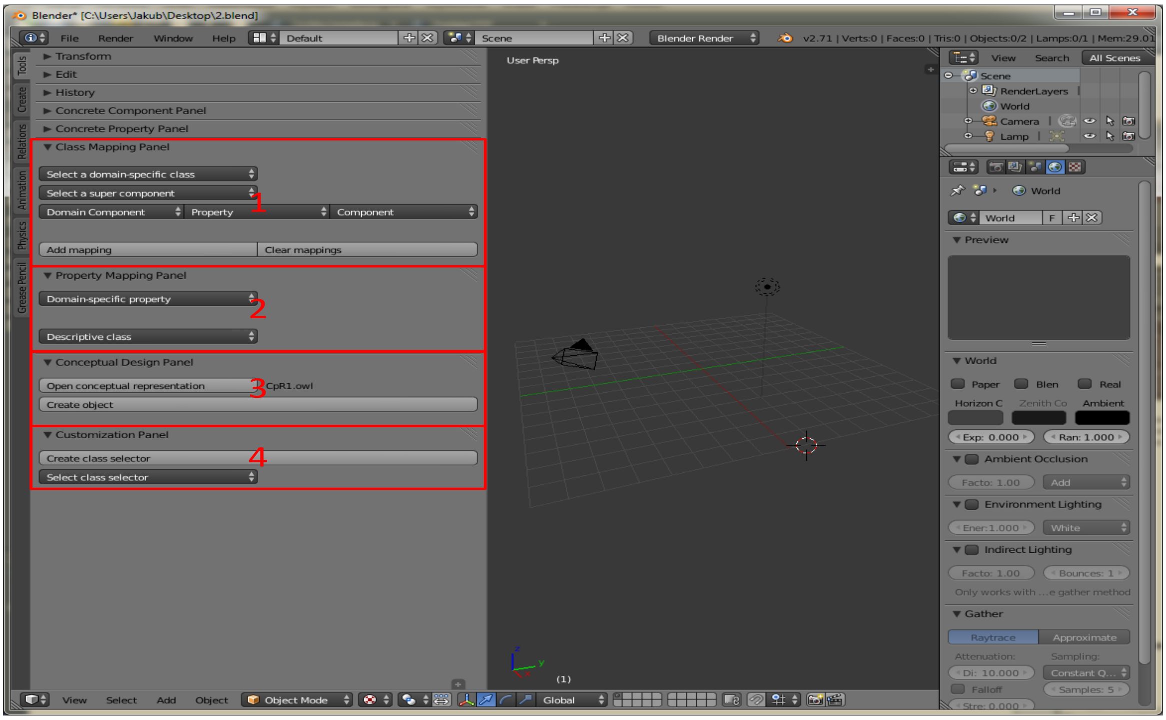


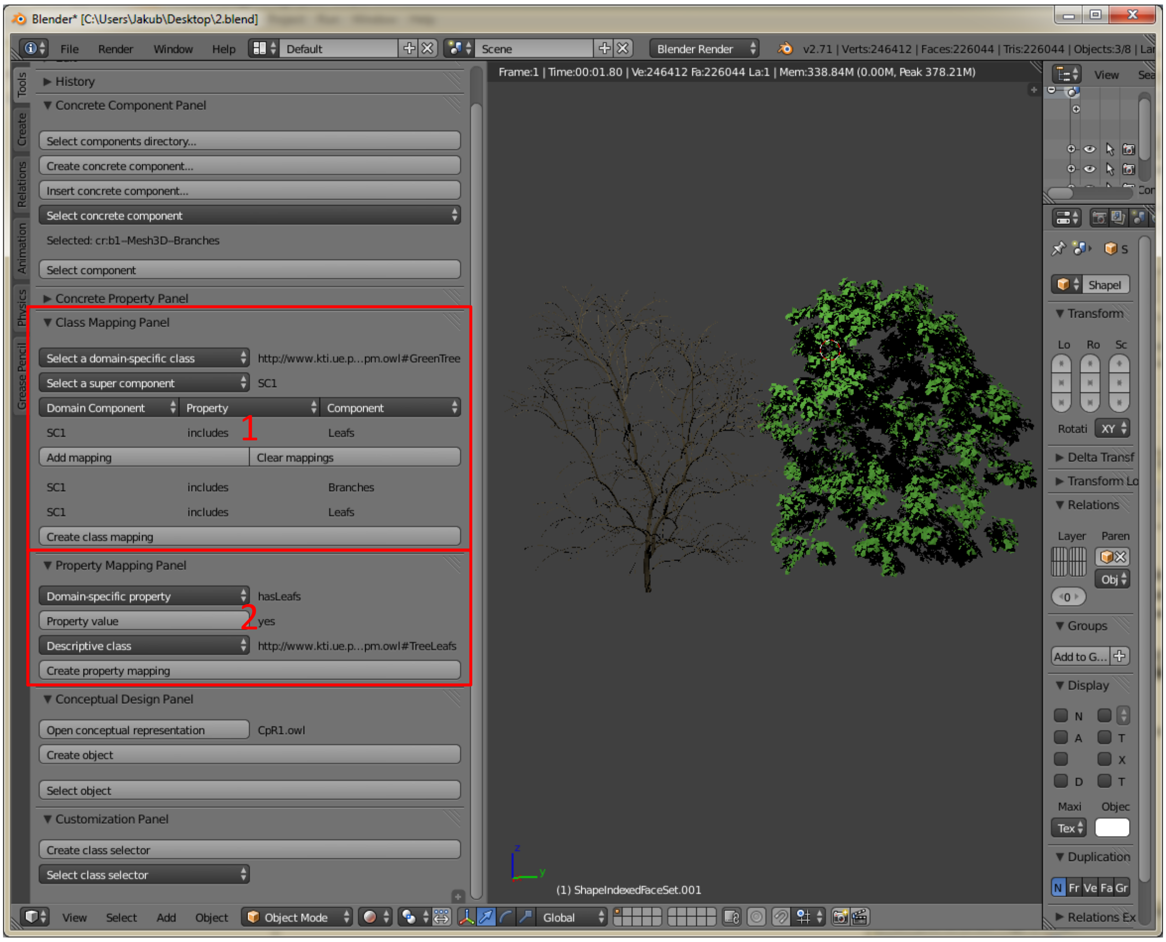
Figure 7.5: The panels of the *Mapping Module* (1,2), *Conceptual Design Module* (3) and *Customization Module* (4)

7.3.1. Creating Descriptive Classes

The creation of a DC requires selection of a domain-specific class to be a DC, selection of a concrete component to be a super-class of the new DC and specification of desirable links (MOPs) between concrete components that will be used for 3D representation of the DC (Figure 7.6/1). For instance, a green tree is a spatial component (a concrete component) that includes branches and leafs (concrete sub-components). Such a specification is encoded as an RM, and it is transmitted to the Knowledge Manager using the service `PUT /<RM>/classes/<domain-specific-class>`. An RM is created according to the MO, in a way that is similar to the creation of a concrete component—using OWL restrictions (as described in Chapter 5). The Knowledge Manager adds the received RM to the overall RM used.

7.3.2. Creating Mapping Data Properties

The creation of an MDP requires selection of a domain-specific data property to become an MDP, specification of a value of the SCM property and specification of a DC that has to be assigned to an object, if the SCM property of the object is set to the value (Figure 7.6/2). For instance, a tree whose `dso:hasLeafs` property is set to `yes` includes both leafs and branches (is a sub-class of the `dso:GreenTree` class), in contrast to a tree with the value of `dso:hasLeafs` set to `no`, which includes only branches. The created RM is transmitted to the Knowledge Manager using the service `PUT /<RM>/properties/<domain-specific-property>`, which adds it to the RM used. The new RM is based on the equivalence between the `owl:hasValue` restriction and the selected domain-specific class (as described in Chapter 5).


Blender* [C:\Users\Jakub\Desktop\2.blend]

Frame:1 | Time:00:01.80 | Ve:246412 Fa:226044 La:1 | Mem:338.84M (0.00M, Peak 378.21M)

v2.71 | Verts:246412 | Faces:226044 | Tris:226044 | Objects:3/8 | La

Concrete Component Panel

Selected: cr:b1–Mesh3D–Branches

Class Mapping Panel

Select a domain-specific class: http://www.kti.ue.p...pm.owl#GreenTree

Select a super component: SC1

Domain Component	Property	Component
SC1	includes	Leafs
SC1	includes	Branches
SC1	includes	Leafs

Property Mapping Panel

Domain-specific property: hasLeafs

Property value: yes

Descriptive class: http://www.kti.ue.p...pm.owl#TreeLeafs

Conceptual Design Panel

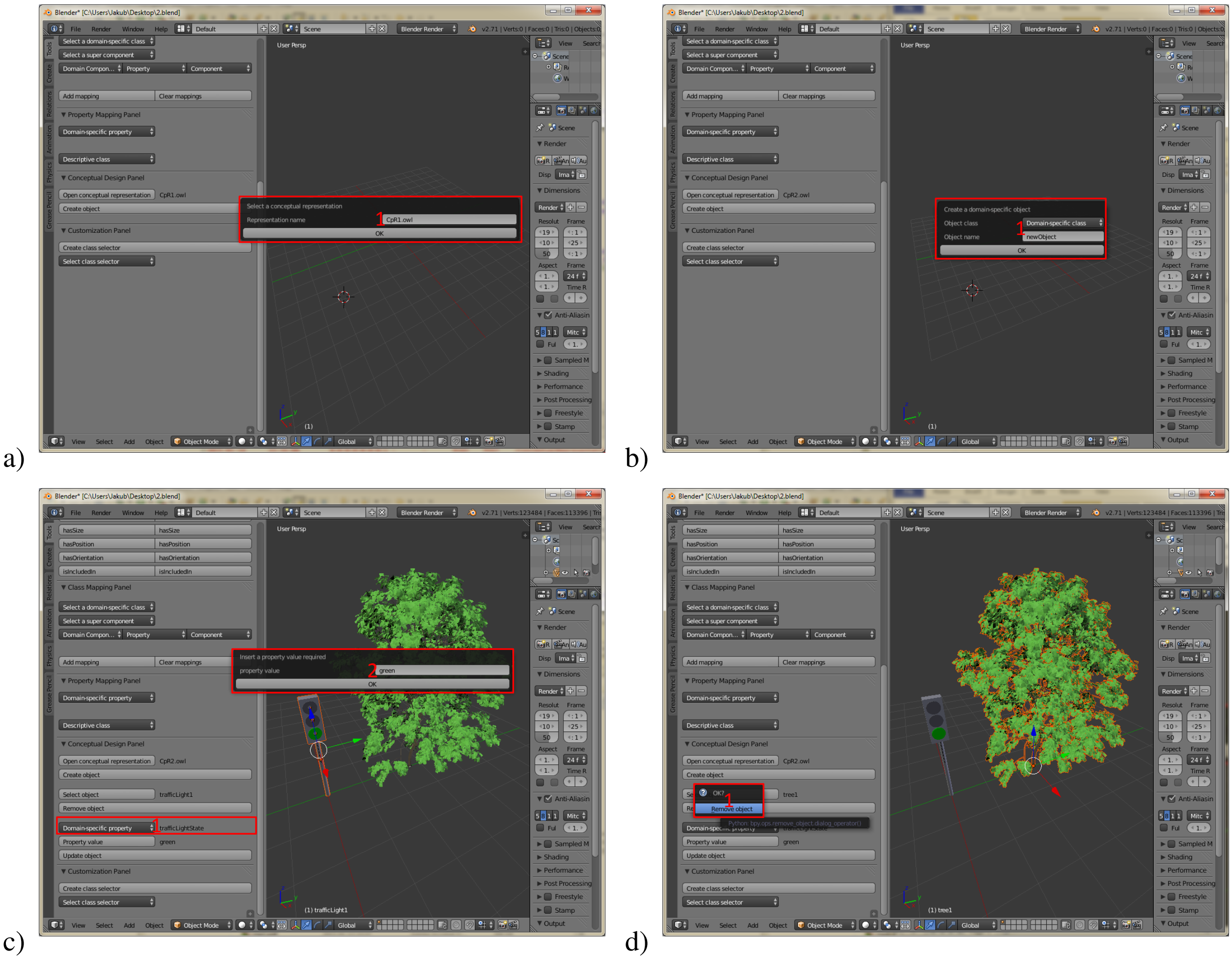
Open conceptual representation: CpR1.owl

(1) ShapeIndexedFaceSet.001

Figure 7.6: Actions available in the *Mapping Module*: creation of a *Descriptive Class* (1) and creation of a *Data Property* (2)

7.4. The Conceptual Design Module

The *Conceptual Design Module* enables accomplishment of Step 3 of SCCM—modeling 3D content with domain-specific concepts (cf. Section 6.4). The GUI of the module consists of the *Conceptual Design Panel* (Figure 7.5/3), which enables the use of DCs and MDPs. The module uses combined representations of domain-specific objects—which are described by both domain-specific and Blender-specific properties. For instance, the `dso:light` domain-specific property determines the current state of a traffic light, while its position in the scene is determined by Blender-specific coordinates, which do not need to have semantic counterparts. In the module, users can create, modify and remove domain-specific objects. The Conceptual Design Module operates on DSOs, CpRs, RMs and CrRs, and it permits the following actions.



Select a conceptual representation
Representation name: CpR1.owl
OK

Create a domain-specific object
Object class: Domain-specific class
Object name: newObject
OK

Insert a property value required
property value: green
OK

Open conceptual representation: CpR2.owl
Select object: trafficLight1
Domain-specific property: trafficLightState
Property value: green

OK?
Remove object
Python: bpy.ops.remove_object.dialog_operator()
Select object: tree1

Figure 7.7: Actions available in the *Conceptual Design Module*: opening a conceptual representation (a), creation of a new domain-specific object (b), modification of a domain-specific object (c) and removal of a domain-specific object (d)

7.4.1. Opening Conceptual Representations

Opening a CpR requires selection of the name of a CpR (Figure 7.7/a/1). The Request Manager uses the Scene Object Manager and the Graphics Engine to clear the current Blender-specific scene. Next, the service `GET /<CpR>` is invoked to receive the requested CpR. Upon receiving the request, the Knowledge Manager combines the CpR with the RM and the CrR to obtain statements on concrete components and concrete properties that represent particular domain-specific objects included in the CpR. The generated PIR is sent to the Client and used to build Blender-specific equivalents of the domain-specific individuals that are inserted into the empty scene.

7.4.2. Creating Domain-specific Objects

The creation of a domain-specific object requires specification of its name and selection of a domain-specific class that has been specified as a DC using the Class Mapping Panel (Figure 7.7/b/1). The list of the available domain-specific classes (included in the DSO) is retrieved using the service `GET /<DSOntology>/classes`. The selected name and class URI are sent to the Knowledge Manager using the service `PUT /<CpR>/individuals/<name>`. The new domain-specific object is added to the CpR. The Knowledge Manager executes the expanding algorithm (cf. Section 6.5) to create a PIR of the domain-specific object with all its concrete components and concrete properties determined by the RM. Finally, the PIR is delivered to the Client and presented using the Graphics Engine.

7.4.3. Manipulating Domain-specific Objects

A manipulation of a domain-specific object requires selection of a Blender-specific object that is a counterpart to the domain-specific object to be modified, as well as a domain-specific property that has been specified as an MDP using the Property Mapping Panel (Figure 7.7/c/1), as well as selection of a value of the MDP, whose assignment will imply presentational effects (Figure 7.7/c/2). The list of the available domain-specific properties (included in the DSO) is retrieved using the service `GET /<DSOntology>/properties`. Modification of a domain-specific object covers several steps, and it relies on the combination of the selected Blender-specific object with its semantic complement determined by the new value of the MDP. First, the RM and the CrR are used by the Knowledge Manager to indicate concrete properties that are determined by the assignment of the new value to the MDP and, therefore, their current values are redundant and will not be respected, as they will be replaced with the new values. The redundant SCM properties are retrieved using the service `GET /<RM>/<property>/<value>`. Second, the PIR of the object is created by the Scene Object Manager, and it encompasses its concrete components and concrete properties without the redundant SCM properties. The PIR is delivered to the Knowledge Manager using the service `PUT /<CpR>/individuals/<object>`. The Knowledge Manager extends the CrR with the concrete components and concrete properties determined by the new value of the MDP. For instance, the assignment of the value `large` to the `dso:size` property results in a specific value of the object scale, therefore the current value of scale may be omitted in the PIR passed to the Server. Finally, the PIR of the object is generated, provided to the Client, converted into the Blender-specific equivalent and used to replace the former Blender-specific object in the scene.

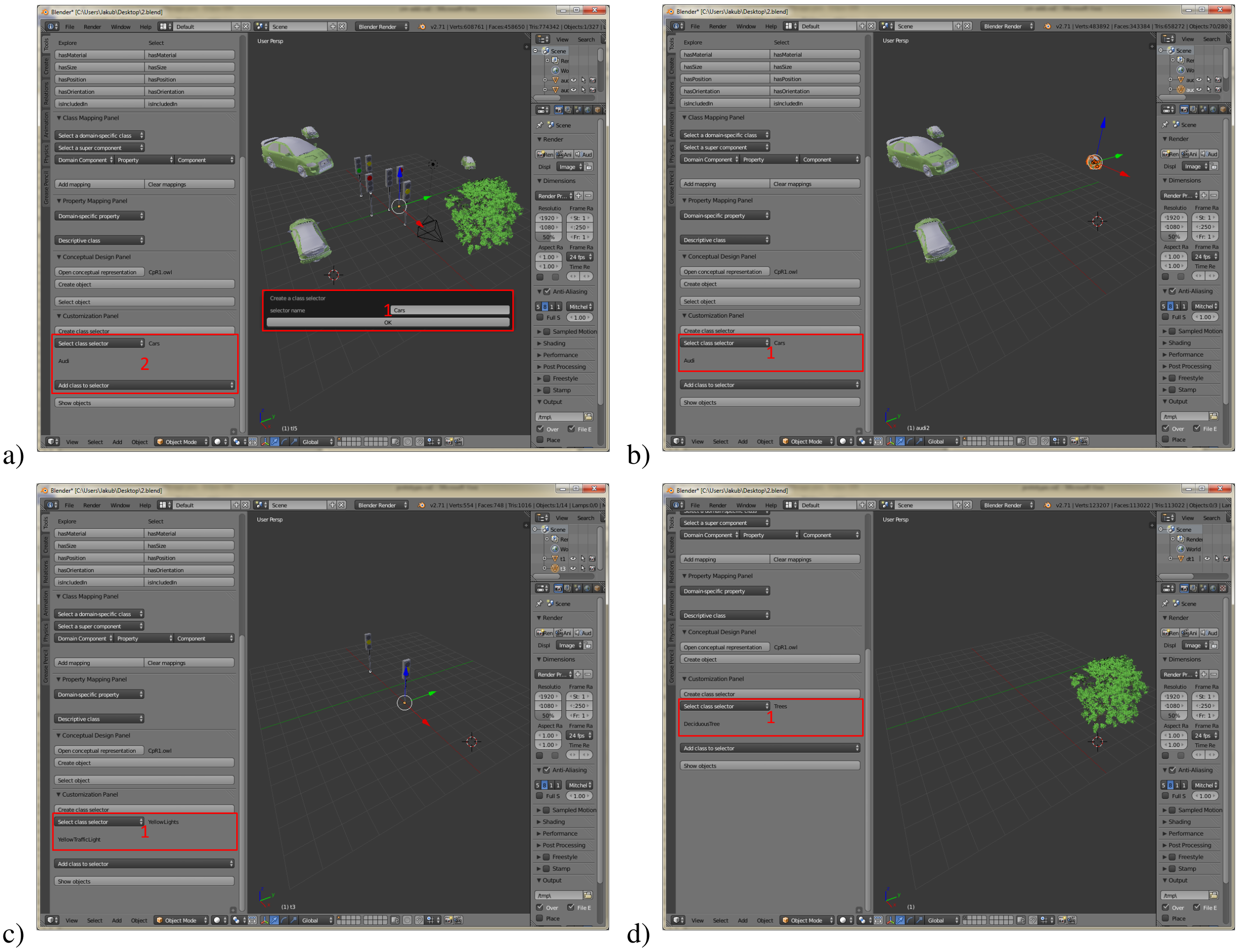
7.4.4. Removing Domain-specific Objects

Removal of a domain-specific object requires selection of a Blender-specific object whose semantic domain-specific counterpart is to be removed from the CpR. The removal of an object is performed by invoking the service `DELETE /<CpR>/individuals/<object>`. Upon receiving the request, the domain-specific object is removed from the CpR by the Knowledge Manager, and its Blender-specific counterpart is removed from the scene by the Scene Object Manager.

7.5. The Customization Module

The *Customization Module* enables accomplishment of Step 5 of SCCM—query-based customization of 3D content based on semantic meta-scenes (cf. Section 6.6). The GUI of the module includes the

Customization Panel (Figure 7.5/4). The module enables a user to perform the sub-class selector CCPs (cf. Section 6.6.2) for on-demand generation of 3D content based on meta-scenes. The sub-class selector CCP results in the creation of an SCM class that permits filtering objects in meta-scenes. The objects that belong to the SCM class are visible in the resulting CtR in contrast to objects that do not belong to the SCM class. For instance, select only cars, yellow traffic lights and trees for presentation, and exclude the other objects from the scene. Query-based generation of 3D content is performed by the Knowledge Manager on the basis of the CO and a 3D meta-scene, which is a PIR.



a) Create a class selector — selector name: Cars; OK. Select class selector: Cars — Audi

b) Select class selector: Cars — Audi

c) Select class selector: YellowLights — YellowTrafficLight

d) Select class selector: Trees — DeciduousTree

Figure 7.8: Actions available in the *Customization Module*: creation of a *sub-class selector* (a) and selection of domain-specific objects with *sub-class selectors* (b,c,d)

7.5.1. Creating Sub-Class Selectors

Creation of an SCM class according to the sub-class selector CCP requires selection of a name (Figure 7.8/a/1) and domain-specific classes (Figure 7.8/a/2) that are to be sub-classes of the created SCM class. The domain-specific classes that are available in the DSO are retrieved using the `GET /<DSOntology>/classes` service and provided to the Customization Panel. The query responsible for the creation of a selector, which includes its name and the list of sub-classes selected by the user, is created using the service `POST /queries/`. The query identifier received in the response is used to perform query-based content customization.

7.5.2. Selecting Domain-specific Objects for Presentation

Selection of domain-specific objects with the sub-class selector results in presentation of only the objects that belong to the created SCM class and its sub-classes (Figure 7.8/b,c,d/1). For instance, the choice of a selector whose sub-classes are green traffic lights and red traffic lights, causes the generation of a scene that includes only such traffic lights. Selection of objects is accomplished using the service `GET /<CpR>/queries/<query-id>`, where `query-id` is received from the response obtained in the creation of the SCM class. The request is processed by the Knowledge Manager, which performs the customization of the meta-scene using the customization algorithm (as described in Section 6.6). The resulting CtR is provided to the Client and visualized by the Scene Object Manager.

7.6. Multi-platform 3D Content Representation Libraries

Java-based libraries have been implemented for generating PSRs according to TKBs. The implementation of the TO and the transformation algorithm is discussed in terms of 3D content representation languages, description of transformation rules and transformation software.

7.6.1. 3D Content Representation Languages

The following languages have been selected for encoding PSRs: ActionScript with the Away3D library, VRML and X3D with XML encoding. PSRs encoded in ActionScript are presented using Adobe Flash Player, while PSRs encoded in VRML and X3D are presented using VRML and X3D browsers, e.g., Cortona3D and Bitmanagement BS Contact, respectively. The languages have been chosen because of the following two reasons. First, they are implemented by a wide range of tools for 3D content presentation. Second, the languages differ in the syntaxes and the approaches to 3D content representation. ActionScript is an imperative language, which permits specification of steps to be accomplished to obtain desirable presentational effects, whereas VRML and X3D are declarative languages, which permit direct specification of desirable presentational effects to be obtained, without specifying steps that must be performed to achieve the effects. Covering different syntaxes and programming paradigms has allowed to achieve more thorough evaluation of the proposed method of generating PSRs.

7.6.2. Transformation Description

TKBs for the selected languages have been implemented using the semantic web standards—the Resource Description Framework (RDF), the Resource Description Framework Schema (RDFS) and the Web Ontology Language (OWL). Similar schemes of VRML and X3D documents have allowed a common TKB for these languages to be developed, while a separate TKB has been developed for ActionScript. SSPts and SLPts are encoded as RDF `bags` and RDF `sequences`, respectively, while the SPts are encoded using RDF reification. TSPts, TPt and TPPts are encoded as instances of appropriate OWL classes and they are linked by OWL properties. Ts, which are included in TBs, are parameterized documents. The signature of a T consists of its name. Ts with the same signatures create a common TS. TPs, which occur within Ts, are indicated by specific symbols. The implemented TKBs and TBs cover the main elements of the ML-SCM, such as shapes and meshes (geometry layer), groups of objects, size, position and orientation (structure layer), textures, materials and light sources (appearance layer) as well as navigation (scene layer). The conformance of the TKBs to the semantic web standards enables transformation of PIRs with a semantic query language (e.g., SPARQL). In contrast to a typical grammar analysis, the use of a query language permits processing of PIRs with regards to complex semantic dependencies between particular statements of the PIR.

7.6.3. The Representation Compiler

A part of the prototype is the *representation compiler*, which is an implementation of the transformation algorithm (cf. Section 6.7). The representation compiler transforms PIRs, which are compatible with the CRO, to PSRs, which are compatible with the selected 3D content representation languages. The representation compiler leverages several software libraries. The *Pellet* OWL reasoner [193] is used in Step 1 of the transformation algorithm to discover implicit statements, which have not been explicitly specified in the processed PIRs, but may be inferred and are necessary for building PSRs. The *Apache Jena* SPARQL engine [78] is used in the next steps of the transformation algorithm to execute queries to PIRs and TKBs when discovering SSs, SLs as well as links between TPts and TPPts.

8. Evaluation of SEMIC

Qualitative and quantitative evaluation has been performed to compare the SEMIC approach with selected approaches to modeling 3D content presented in Chapters 2 and 3 and to discuss the characteristics of SEMIC.

8.1. Qualitative Evaluation

The SEMIC approach has been compared to selected approaches to modeling 3D content in terms of functionality. The analysis aims to indicate the major gaps in the available approaches to 3D content creation, which are covered by SEMIC. The following solutions designed for 3D content creation have been analyzed.

- The available approaches to semantic modeling of 3D content, which enable 3D content creation according to the declarative paradigm, such as the approaches proposed by Latoschik et al., Troyer et al. and Kalogerakis et al. presented in detail in Chapter 3.
- Languages that are widely-used for imperative programming of web applications with selected libraries designed for 3D content creation, such as ActionScript with Away3D and Java with Java3D.
- Widely-used applications for visual modeling of 3D content:
  — domain-specific modeling environments that have been intended to facilitate 3D content creation by domain experts (e.g., architects, interior designers, engineers, etc.) without advanced technical skills, by providing a number of intuitive and easy-to-use modeling tools, e.g., enabling the creation of shapes, basic animations, interactions, etc.
  — generic modeling environments that have been intended for modeling 3D content by professional content developers, providing a number of comprehensive features, such as extensibility and scripting languages.

  SketchUp and 3DVIA have been selected in the first category, while Blender and 3ds Max have been selected in the second category. These solutions are well-documented and they are used by large communities, therefore, a number of internet fora, articles as well as textual and video tutorials are attainable for them.

The analysis covers several aspects that are significant in the context of content creation and content use in modern 3D/VR/AR applications (Figure 8.1).

8.1.1. Aspects of 3D Content

The primary aspects of 3D content creation, such as modeling basic and complex geometrical shapes, appearance, structural objects, animations, interactions, complex behavior and physics, are covered by the majority of the analyzed approaches to declarative, imperative and visual 3D content creation. Both the imperative languages and the visual environments enable imperative scripts, which can express arbitrarily complex content behavior. In contrast to them, the declarative approaches permit at most

Modeling paradigm	**Declarative content creation**				**Imperative content creation**	**Visual content creation**	
Criterion \ Approach	**SEMIC**	**Latoschik et al.**	**Troyer et al.**	**Kalogerakis et al.**	**ActionScript (Away3D) Java (Java3D)**	**SketchUp, 3DVIA**	**Blender, 3ds Max**
Addressed aspects of content creation							
modeling geometry	-	-	-	-	-	x	x
modeling basic geometrical shapes	x	x	x	x	x	x	x
modeling appearance	x	x		x	x	x	x
creating structural objects (assemblies)	x	x		x	x	x	x
creating animations	x	x	x	x	x	x	x
describing interactions	x	x	x	x	x	x	x
modeling complex behavior	-	o	o		x	x	x
modeling physics	-	x	x	-	x	x	x
Conceptual content creation							
using concepts directly related to 3D computer graphics	x	x	x	x	x	x	x
using abstract concepts (nonspecific to 3D computer graphics)	x	x	x		x	-	x
using hierarchies of classes	x	x	x	x	x	-	x
using hierarchies of properties	x	x			x	-	x
representation of complex features by classes	x	x	x		x	-	x
representation of complex features by properties	x				x	-	x
representation of complex features by individuals	x				x	-	x
representation of complex features by combinations of classes	x				-	-	x
representation of complex features by combinations of properties	x				x	-	x
representation of complex features by combinations of individuals	x				x	-	x
representation of relations between objects	x				x	-	x
representation of complex features by rules	x			x	-	-	-
specifying schemes of scenes	x		x		x	-	x
separation of concerns in content creation	x						
specifying compatibility between objects	x		x	x	x	-	x
Knowledge-based content creation							
content modeling based on constraints	x	x	o	o	-	-	o
discovering object properties on the basis of object classes	x				o	-	-
discovering object classes on the basis of object properties	x				-	-	-
discovering dependencies between objects on the basis of object classes and object properties	x				-	-	-
modifying objects on the basis of properties of other objects	x			x	x	-	x
Multi-platform content creation							
generating final content representations for different platforms	x	o			-	x	x
generic description of content transformation	x				-	-	-
possible introduction of new presentation platforms	x				-	-	x
Support for the use in content repositories							
convenient for indexing content	x	x	x	x	-	-	-
convenient for searching content	x	x	x	x	-	x	-
convenient for managing individual objects	x	x	x	x	-	-	-
convenient for managing groups of objects	x				-	-	-
creating content with a query language	x				-	-	-
managing content with a query language	x			o	-	-	-
convenient for exploring content with a query language	x	x	x	x	-	-	-
reusing common concepts	x		x	x	x	x	x

‘x’ – the solution meets the criteria

‘○’ – the solution does not fully meet the criteria

‘–’ – the solution does not meet the criteria

‘ ’ – the information is not available

Figure 8.1: Comparison of selected approaches to modeling 3D content

limited modeling of behavior based on specific sets of concepts, such as templates of particular animation types (e.g., walking). The SO-SEMIC environment does not currently enable modeling of complex content behavior and physics. However, appropriate extensions of 3D content to 3D scenarios including declarative descriptions of arbitrarily complex content behavior are planned for SO-SEMIC.

8.1.2. Conceptual 3D Content Creation

Conceptual content creation has been considered in terms of representation of 3D content at different levels of abstraction (detail) and the use of the well-established semantic web concepts (such as classes, individuals, properties and rules) in the 3D content creation process. Overall, the available declarative approaches enable the use of basic semantic expressions (combinations of semantic concepts), such as classes and properties at different levels of abstraction in modeling of content. However, they do not permit a number of more sophisticated combinations of concepts, which are essential to visualization of complex knowledge bases and which are covered by SEMIC (marked in blue in the table). The imperative languages and generic modeling environments permit complex conceptual content representation at any level of abstraction, however, expressed imperatively, which is not convenient for knowledge discovery and content management in web repositories. The available approaches do not support separation of concerns between different users, who have different modeling skills and experience and are equipped with different modeling tools.

8.1.3. Knowledge-based 3D Content Creation

Knowledge-based 3D content creation has been considered in terms of building content representations with regards to discovered properties and dependencies of content objects, which may be hidden (not explicitly specified), but which are the logical implications of facts that have been explicitly specified in the knowledge base. On the one hand, this aspect of content creation is not available in imperative 3D content representation languages, including the languages used in the generic modeling environments, as in these approaches, knowledge discovery must be implemented in advance. On the other hand, although the available declarative approaches could be extended to enable knowledge-based modeling, currently, they do not address content creation based on discovered data. In contrast to the other approaches, SEMIC enables discovery of knowledge from various combinations of statements and rules, and visualization of the inferred facts.

8.1.4. Multi-platform 3D Content Creation

Although, the analyzed visual environments support different content formats and the generic modeling environments (Blender and 3ds Max) enable the introduction of new formats (e.g., by implementing appropriate plug-ins), they do not enable generic description of content transformation that is independent of particular presentation platforms. Such description could facilitate the introduction of new content formats. Overall, the declarative approaches do not permit flexible and generic content transformation for different platforms. In SEMIC, the description of content transformation is generic—independent of particular 3D content presentation platforms. Only the rules of transformation must be individually specified for particular platforms depending of the 3D content representation language used.

8.1.5. The Use of 3D Content in Content Repositories

Due to the use of semantic techniques, the declarative approaches to content creation could be potentially used in 3D content repositories to facilitate content creation and management using ontologies and knowledge bases. However, the available declarative approaches do not enable content creation and management with semantic query languages (e.g., SPARQL), which is an important shortcoming in the context of the use of content in repositories. Overall, the generic modeling environments have not been intended to be used together with shared content repositories. The domain-specific modeling environments (SketchUp and 3DVIA) support basic searching for 3D models in repositories and retrieval of 3D models from repositories. In contrast to the other approaches, SEMIC enables query-based 3D content creation and management at different levels of abstraction (both concrete and conceptual).

8.2. Quantitative Evaluation

The quantitative evaluation covers different elements of SEMIC: the SCM and SCCM (including the use of CCPs) with the focus on representations created by users and activities performed by users as well as the algorithms, for which computational complexity and performance have been analyzed. The elements are related to different aspects of 3D content creation and representation. The quantitative evaluation has been conducted using the implemented SO-SEMIC environment.

8.2.1. The Semantic 3D Content Model

The evaluation of the SCM covers CpRs, PIRs and PSRs. The CpRs and PIRs have been encoded with SCM using the RDF-Turtle format. The DSO used to build the CpRs provides different types of complex objects and complex properties combining multiple concrete components of 3D content. PSRs have been encoded with the VRML, X3D and ActionScript (with the Away3D library) languages.

The evaluation has been carried out starting with CpRs assembled from different numbers of objects. The number of objects has varied over the range of 5 to 50 with the step equal to 5. For every number of objects, 20 scenes have been randomly generated and the average results have been calculated. The test environment used was equipped with the Intel Core i7-2600K 3.4GHz CPU, 8 GB RAM and the Windows 7 OS.

The Size and Complexity of 3D Content Representations

The complexity of 3D content representations has been evaluated with the following metrics: the Structured Document Complexity Metric [187], the number of bytes, the number of logical lines of code (LLOC) [72] and the Halstead Metrics [146]. The first metric has been used to measure the complexity of representation schemes of CpRs, CrRs and PSRs, whereas the other two metrics have been used to measure the complexity of particular content representations.

Structured Document Complexity Metric. The Structured Document Complexity Metric has been used to measure the complexity of representation schemes regarding unique elements and attributes, required elements and attributes as well as attributes that need to be specified at the first position within their parent elements. The metric may be calculated for grammar-based documents. The values of the Structured Document Complexity Metric calculated for the schemes of CpRs, CrRs and PSRs (encoded in VRML, X3D and ActionScript) are presented in Table 8.1.

Criteria	VRML/X3D	AS	PIR	CpR	RM
Unique elements	15	24	24	7	36
Unique attributes	21	27	3	14	8
Required elements	1	1	12	4	4
Required attributes	8	5	3	2	2
Elements at position 1	0	0	0	0	0
Sum	45	57	42	27	50

Table 8.1: Structured Document Complexity Metric of 3D content representations

The results obtained for VRML and X3D representations are equal, because both languages use schemes with equivalent basic elements and attributes. While in VRML and X3D representations, different hierarchical document elements have been classified as unique elements, in ActionScript representations, different classes and data types (potentially corresponding to different elements of VRML/X3D) have been classified as unique elements. In PIRs, unique elements cover different RDF, RDFS and OWL elements as well as semantic properties of 3D content, which are encoded by document elements (according to the RDF syntax). In comparison to PIRs, CpRs include a lower number of unique elements, which are semantic combinations of multiple elements of PIRs. Different properties occurring in hierarchical VRML/X3D elements or properties of objects in ActionScript representations have been classified as unique attributes in the metric calculation. Since, in PIRs the content properties are encoded using document elements, only a few attributes, which are primary RDF, RDFS and OWL attributes, may be classified as unique attributes in PIRs. In comparison to PIRs, CpRs have a higher number of unique attributes, which is determined by the DSO used. RMs incorporate the highest number of unique elements of all analyzed documents, because they indicate unique elements of both concrete and conceptual representations. Unique attributes of RMs are semantic properties specified in the MO. In VRML/X3D and ActionScript representations, the `scene` and the `view` are the only required elements, and they have a few required attributes. The elements and the attributes that are required in CpRs, PIRs and RMs, are basic RDF, RDFS and OWL elements and properties. The calculated values of the Structured Document Complexity Metric show that the overall complexity of CpRs is much lower than the overall complexity of PIRs, which is a little lower than the overall complexity of VRML/X3D and much lower than the overall complexity of ActionScript representations. The overall complexity of RMs is the highest of all the complexities calculated.

Size Metrics. The number of bytes and the number of logical lines of code (LLOC)—without comments—have been used to measure the size of content representations (Figure 8.2). The graphs present the metrics in relation to the number of domain-specific components included in CpRs. The tests confirm the O(n) space complexity of semantic content representations created using the SCM, where n is the number of components in the representations.

The differences in size are relatively high between different types of representations. In both comparisons, CpRs are more than twice more concise than the corresponding PSRs, as the CpRs are transformed to multiple concrete content components. PIRs are most verbose, which confirms that RDF-turtle is the most verbose encoding format of all the encoding formats used, taking into account that the elements of PIRs are semantic equivalents to the elements of the corresponding PSRs.

Halstead Metrics. The Halstead metrics have been used to measure the complexity of content representations in several aspects: the size of vocabulary and length of the content representations, volume corresponding to the size of the representations, difficulty corresponding to the error proneness

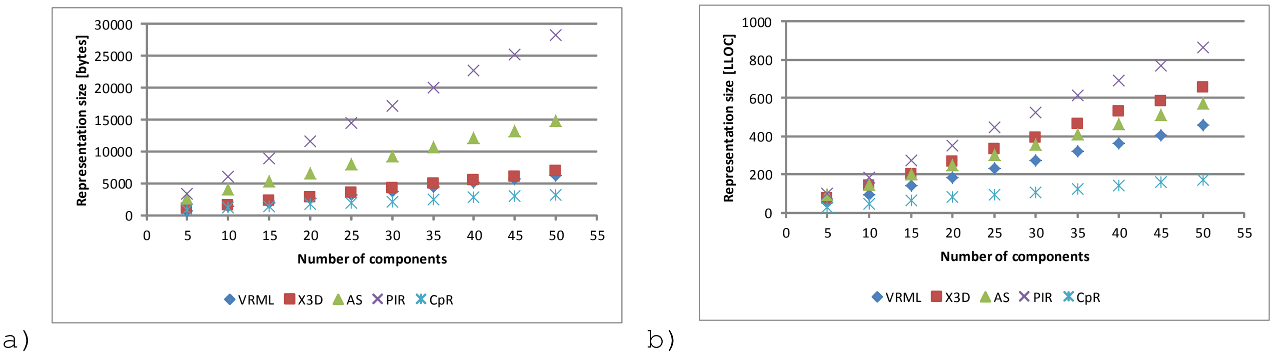


Figure 8.2: The size of representation depending on the number of objects—in bytes (a) and in LLOC (b)

of the representations, effort in implementation and analysis of the representations as well as estimated time required for the development of the representations. The particular Halstead metrics in relation to the number of objects of CpRs are presented in Figures 8.3-8.8. The VRML and X3D representations that have been considered in the presented evaluation are based on equivalent basic elements and attributes, therefore they have been presented together.

Vocabulary (Figure 8.3), which is the sum of unique operators (*n1*) and unique operands (*n2*) of the representation:

$$Voc = n1 + n2 \tag{8.1}$$

is lowest for CpRs and highest for VRML/X3D, because of a high number of unique operands, which are individual scene graph nodes. In contrast to the other languages, in VRML/X3D, a relation between two components in the generated representation is reflected by nesting one component in another component with specifying all intermediate nodes, which are also classified as unique operands, e.g., applying a `Material` to a `Shape` requires an intermediate `Appearance` node to be nested in the `Shape` node. In the other languages, such associations are usually described directly—without any intermediate nodes.

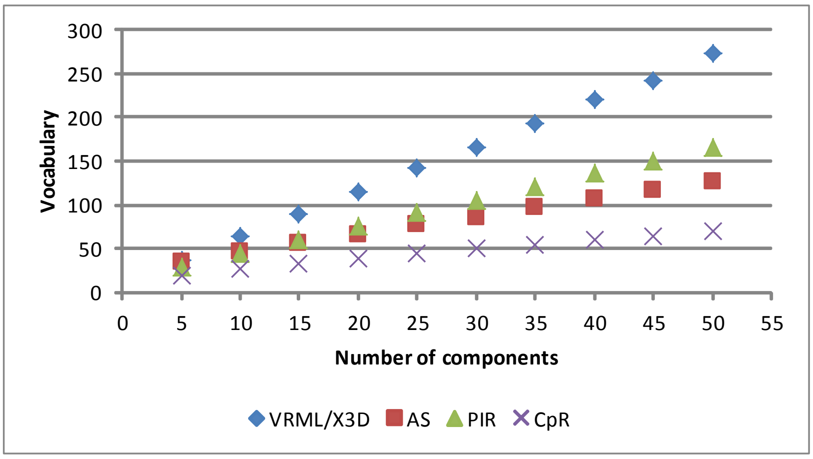


Figure 8.3: The vocabulary of representation depending on the number of objects

Length (Figure 8.4), which is the sum of the total number of operators (*N1*) and the total number of operands (*N2*) of the representation:

$$Len = N1 + N2 \tag{8.2}$$

is lowest for CpRs. VRML/X3D representations predominate PIRs and ActionScript representations, as their operands typically occur once and all references to them are specified by nesting attributes and other nodes in them. Therefore, the operands do not require to be additionally explicitly indicated (e.g., by proper instructions or statements), and thus the length of VRML/X3D representations is lower than the length of the other representations, in which all references to operands must be explicitly declared by referring to their identifiers.

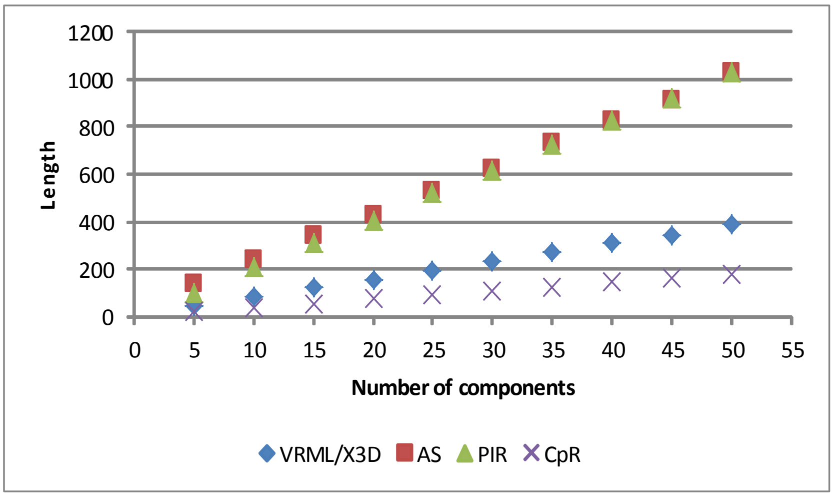


Figure 8.4: The length of representation depending on the number of objects

The graph of volume (Figure 8.5), which depends on the length and vocabulary of content representations:

$$Vol = Len * log_2(Voc) \tag{8.3}$$

is similar to the graph of length for all types of representations.

In contrast to the other Halstead metrics discussed, difficulty (Figure 8.6), which is given by the formula:

$$Diff = \frac{n1}{2} * \frac{N2}{n2} \tag{8.4}$$

has similar values for different numbers of components in the scene. It is lowest for conceptual and VRML/X3D representations (low error proneness) because of the relatively low number of unique operators and the total number of operands as well as the relatively high number of unique operands. Relatively high difficulty of ActionScript representations (high error proneness) is caused by relatively high values of the first two factors and the relatively low value of the third factor.

Effort (Figure 8.7) and time (Figure 8.8) required by a user to implement or analyze a representation, which are the products of its difficulty and volume:

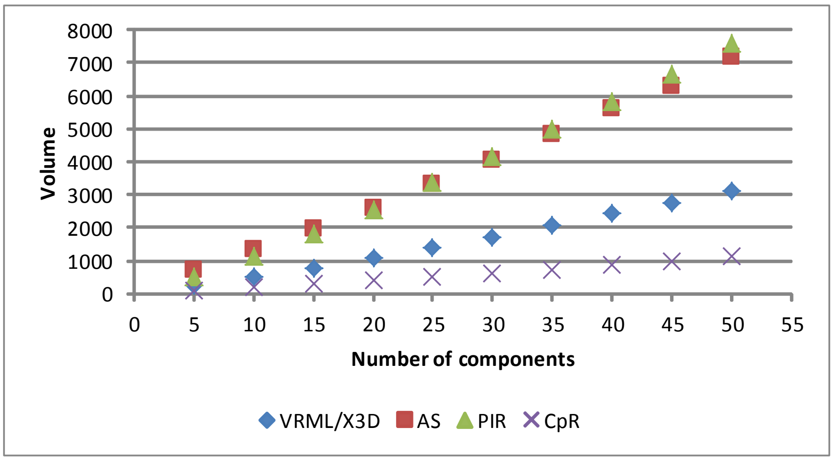


Figure 8.5: The volume of representation depending on the number of objects

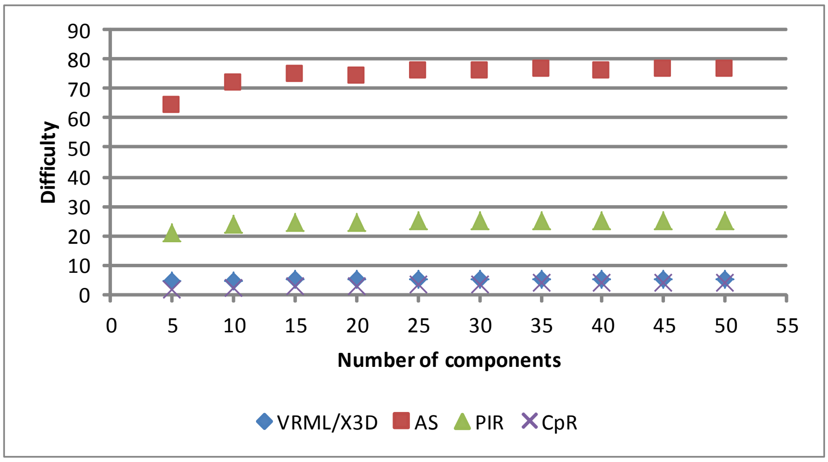


Figure 8.6: The difficulty of representation depending on the number of objects

$$Eff = Diff * Vol \tag{8.5}$$

$$Time[h] = \frac{Eff}{18 * 3600} \tag{8.6}$$

are lowest for CpRs because of the relatively low difficulties and volumes. Several times higher values of effort and time occur for the other representations.

Multi-platform 3D Content Representation

The capability of generating multi-platform 3D content representations has been evaluated in terms of the following four metrics:

- the profit from automatic generation of platform-specific representations;
- the cost of elementary changes in platform-specific representations;

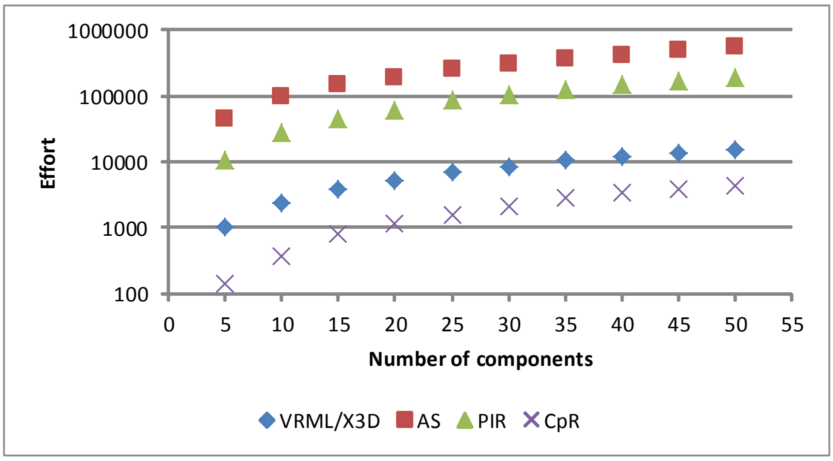


Figure 8.7: The effort in analyzing the representation depending on the number of objects

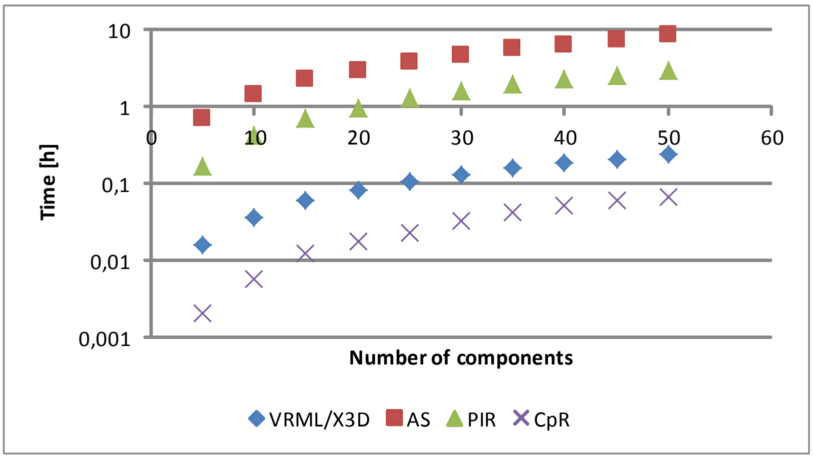


Figure 8.8: Time of implementing the representation depending on the number of objects

- the cost of the introduction of a 3D content presentation platform;
- the profit from code generation for a new 3D content presentation platform.

The profit from automatic generation of platform-specific representations. The profit from automatic generation of PSRs for different platforms in relation to the number of objects of the primary representation (a PIR) has been calculated as the ratio of the overall size of the PSRs (encoded in VRML, X3D and ActionScript), which have been generated on the basis of the primary PIR (implemented with the CRO) and encoded with the selected 3D content representation languages, to the size of the PIR:

$$Profit = \frac{Size(PSRs)}{Size(PIR)} \tag{8.7}$$

The metric shows, how much work on implementation can be saved when implementing a PIR and using the proposed method to automatically generate the corresponding PSRs, rather than implementing the desirable PSRs from scratch using the particular languages, independently one from another. The profit is presented in Figure 8.9.

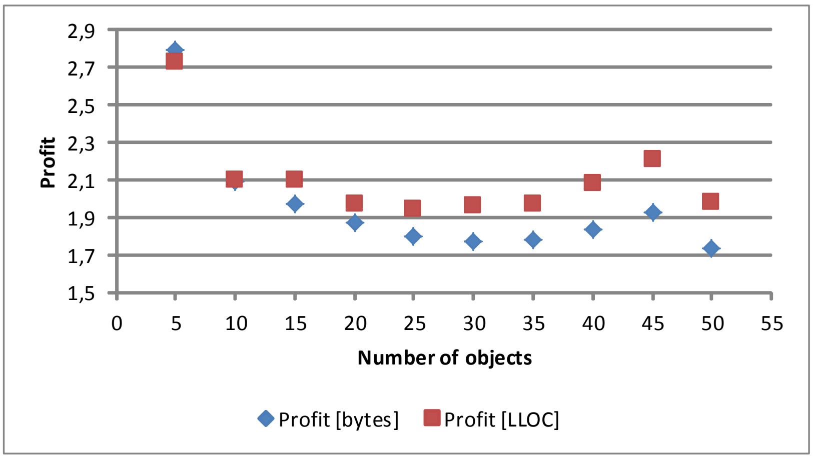


Figure 8.9: The profit from automatic generation of *platform-specific representations*

The values of the profit vary predominantly in the range 1.7 to 2.1—for the size expressed in the number of bytes—and from 1.9 to 2.3—for the size expressed in the number of LLOC. Although the profit values are decreased by the relatively high size of PIRs in comparison to the corresponding PSRs, about 50% of work may be saved when using the proposed solution with the three presentation platforms. A higher number of presentation platforms would result in higher values of profit.

The cost of elementary changes in platform-specific representations. The cost of elementary changes in content representations is the average size of code that is required to be added, deleted or modified to add, delete or modify a component or a property of a 3D content representation. It is assumed that a change of a representation is related to at least one of its basic elements. A basic element of a PIR is an S, while a basic element of a PSR that corresponds to the PIR is a TS associated with an S from the PIR. Hence, the cost of an elementary change in a content representation is the average size of an S (for PIRs) or the average size of a TS (for PSRs):

$$Cost = AvgSize(S|TS) \tag{8.8}$$

The values of the metric have been calculated for the selected 3D content representation languages. The results are presented in Table 9.2 (in bytes and LLOC).

Size	VRML	X3D	ActionScript	CRO
NoB	35	39	130	39
LLOC	2	2	4	1

Table 8.2: The cost of elementary changes in content representations

In SEMIC, Ss of the CRO typically gather multiple instructions (multiple LLOC) of the other 3D content representation languages. ActionScript is the most verbose of the selected languages, as to represent some aspects of 3D content on the Flash platform (e.g., navigation), a number of LLOC need to be implemented to create an appropriate T. The most concise language is VRML, which is equivalent to X3D in terms of the provided level of abstraction. A slight difference between VRML and X3D in the number of bytes is caused by the difference in their syntaxes—the syntax of VRML is more concise than the verbose XML-based syntax of X3D chosen for the evaluation.

The cost of the introduction of a 3D content presentation platform. The cost of the introduction of a content presentation platform is the overall size of code that must be implemented to introduce a 3D content presentation platform into the system. Hence, the cost incorporates the size of the TKB, which describes the transformation of PIRs to PSRs, and a TB, which corresponds to the language of the new platform introduced:

$$Cost = Size(TKB) + Size(TB) \tag{8.9}$$

The metric has been calculated and expressed in the number of bytes and the number of LLOC for the selected 3D content representation languages that determine the content presentation platforms to be used (Table 9.3).

	VRML		X3D		ActionScript	
	NoB	LLOC	NoB	LLOC	NoB	LLOC
TKB	22291	789	22291	789	25290	943
TB	1193	71	1312	71	4957	152
Sum	23484	860	23603	860	30247	1095

Table 8.3: The cost of the introduction of a content presentation platform

On the basis of the calculated values of the metric, an estimated cost of the introduction of a new platform into the system may be determined. A new platform, whose functionality (a set of content components and properties) needs to be equivalent to the already implemented platforms, is anticipated to require about 860-1095 LLOC. The lower bound is more probable for hierarchical declarative languages (e.g., XML3D), whereas the upper bound is more probable for structural and object-oriented imperative languages (e.g., Java3D). However, the exact value depends on the capabilities of a particular target language (and programming libraries) used as well as the complexity of the CRO. The more advanced the capabilities of the language and the libraries provided, the lower the cost should be, e.g., a single instruction may enable one of several available navigation modes.

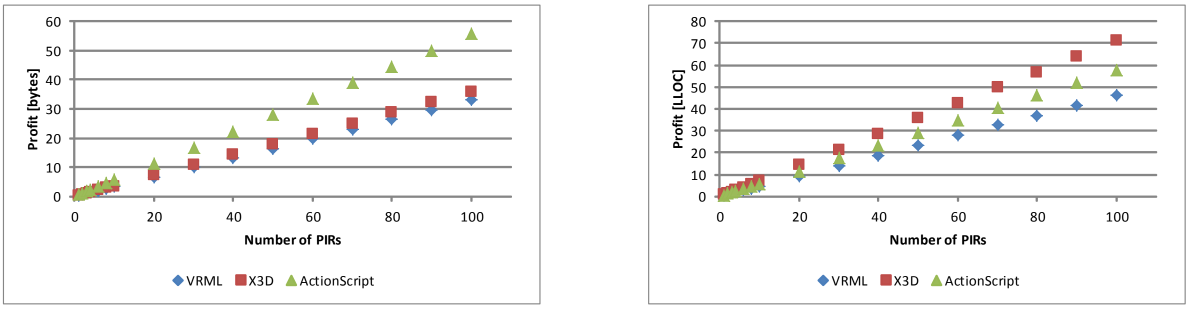
The profit from code generation for a new 3D content presentation platform. The profit from code generation for a new platform is directly proportional to the number of PIRs that are available in the system and that are to be transformed to PSRs compliant with the new platform, and the size of the generated PSRs. The metric is given by the formula:

$$Profit = \frac{N * Size(PSRs)}{Cost(TKB + TB)} \tag{8.10}$$

where: *N*—the number of PIRs that are available in the system and are to be transformed; *Size(PSRs)*—the size of PSRs to be generated; *Cost(TKB + TB) = Size(TKB) + Size(TB)*—the overall size of the TB and the TKB that enable the transformation (calculated in the previous section).

The metric has been calculated for a PIR including 50 objects. However, for larger scenes the profit from transformation could be higher. The values of the profit in relation to the number of PIRs are presented in Figure 8.10.

The profit increases linearly with the increase in the number of PIRs that are available in the system and need to be transformed. The attainable profit is higher for the X3D and ActionScript languages, as the size of X3D and ActionScript representations is typically larger than the size of VRML representations.



a) b)

Figure 8.10: The profit from code generation for a new platform: for PSR size in bytes (a) and for PSR size in LLOC (b)

8.2.2. The Semantic 3D Content Creation Method

SCCM has been evaluated in terms of user's effort in creation and customization of 3D content in the visual environment. The efficiency of content creation and customization has been measured with three metrics: the *number of keystrokes and mouse clicks*, *mouse distance* and *time* required to accomplish modeling activities. While, in the evaluation, the number of keystrokes and mouse clicks is representative mainly for measuring the complexity of the user interface (e.g., the design of and links between windows and controls), mouse distance is representative mainly for measuring the complexity of manual operations performed on the modeled content (e.g., zooming a view, dragging and dropping content elements, etc.). Time covers the entire modeling process—both activities performed by the user and the processing of content performed by the environment. The metrics have been measured for the creation and customization of the following four 3D scenes.

1. *Scene 1*: Crossroads with cars, which have low and high beams (Figure 8.11).
2. *Scene 2*: Crossroads with cars, some of which are priority vehicles (Figure 8.12).
3. *Scene 3*: Crossroads with traffic lights, which are in different states (Figure 8.13).
4. *Scene 4*: A housing with trees, which have different scales and colors of leafs (Figure 8.14).

In the scenes, the models [27]- [39] have been used.

Every scene has been individually modeled 10 times by a user using the Blender and SO-SEMIC modeling environments, and the average results have been calculated for every metric and every environment. Further, *average gain* has been calculated for every metric as the average ratio of the result obtained in Blender to the result obtained in SO-SEMIC. The modeling environments offer different capabilities and are characterized by different degrees of complexity of 3D content creation. Blender is a complex modeling environment with a number of advanced functions directly related to 3D computer graphics, requiring relatively high skills in 3D content creation. SO-SEMIC is an environment enabling creation of 3D content by referring to high-level meaning of particular content components, which makes 3D content creation more intelligible to non-IT-specialists. The tests were performed using a computer equipped with the Intel Core 2 Duo CPU 2.53GHz, 4GB RAM, NVIDIA GeForce 9500M and Windows

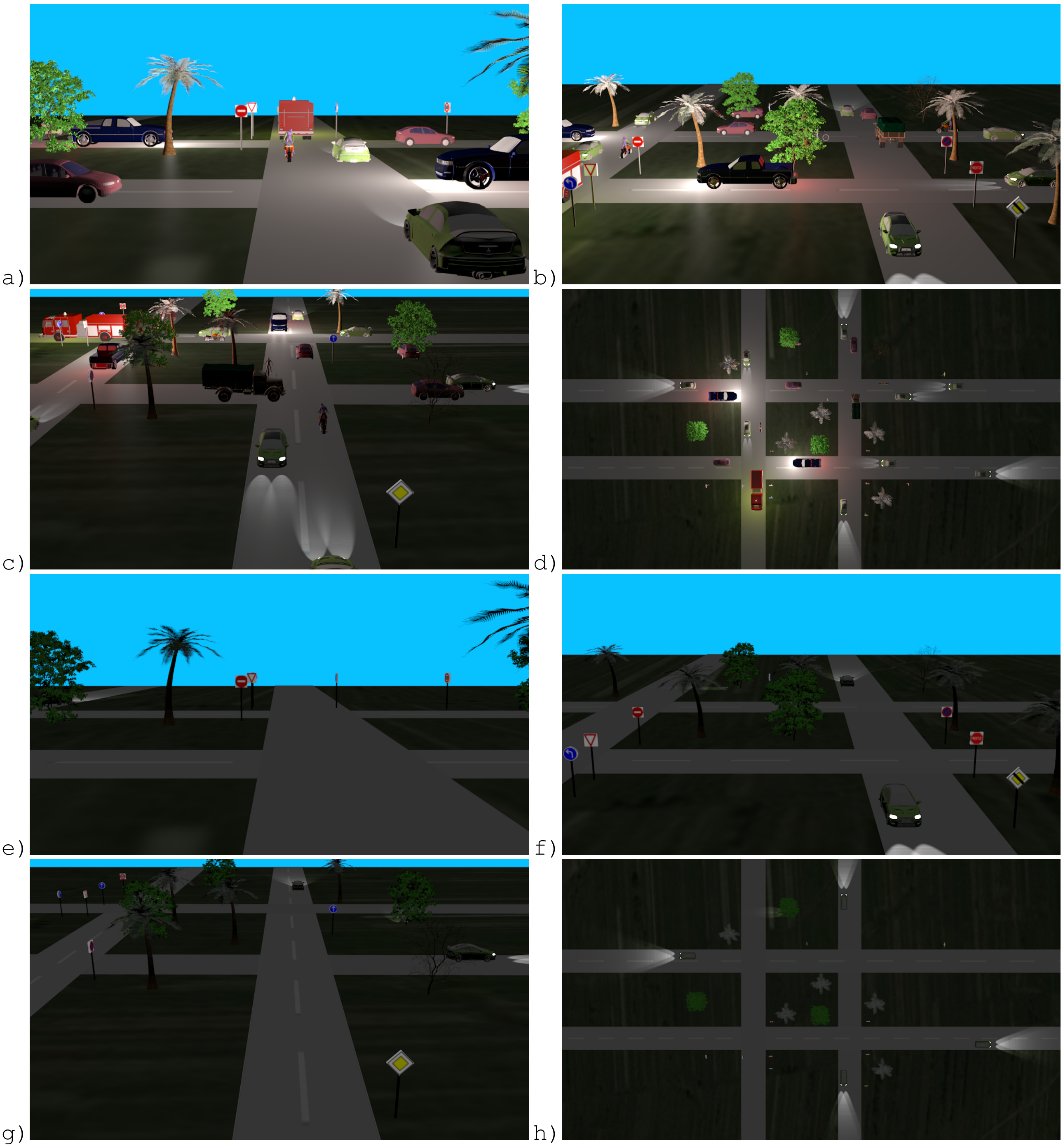


Figure 8.11: Crossroads with cars seen from different points (a-d), only cars with high beam are selected for presentation (e-h)

7 Professional. On the computer, both modeling environments—Blender 2.71 as well as SO-SEMIC (including the Client and the Server)—have been installed.

Designing Conceptual 3D Content Representations

The average values of the metrics have been calculated for the creation of the 3D scenes (Table 8.4). The metrics depend on the number of objects in the scene, their structural complexity as well as the number of SCM object properties set while creating the scene. In addition, time required for modeling depends on the geometrical complexity of the objects, since the rendering of more complex shapes

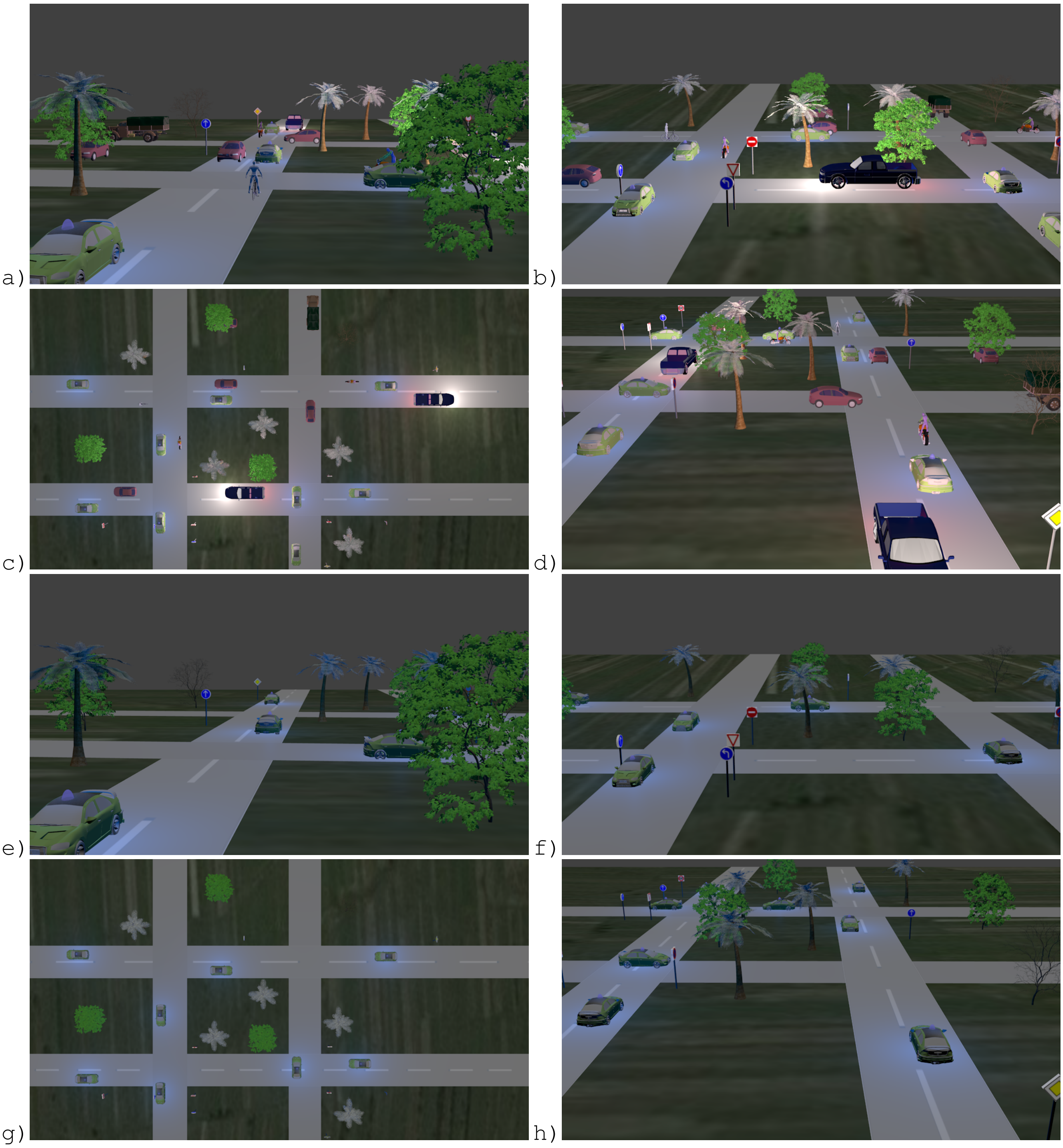


Figure 8.12: Crossroads with cars seen from different points (a-d), only priority vehicles are selected for presentation (e-h)

requires more computations. The number of objects is the lowest in Scene 2 and the highest in Scene 3, while the structural and geometrical complexity of objects is the lowest in Scene 2 (relatively simple models of traffic signs and traffic lights) and the highest in Scene 4 (relatively complex models of trees and houses). Modeling content with Blender requires almost four times more keystrokes and mouse clicks then modeling content with SO-SEMIC. The high value of average gain is caused by the possibility of determining a number of object features by setting single domain-specific properties in SO-SEMIC. For instance, the color of leafs of a tree and their relative position to the branches are determined by setting the `leafs` SCM property to `green` or `yellow`. Such capability is not available in Blender, in

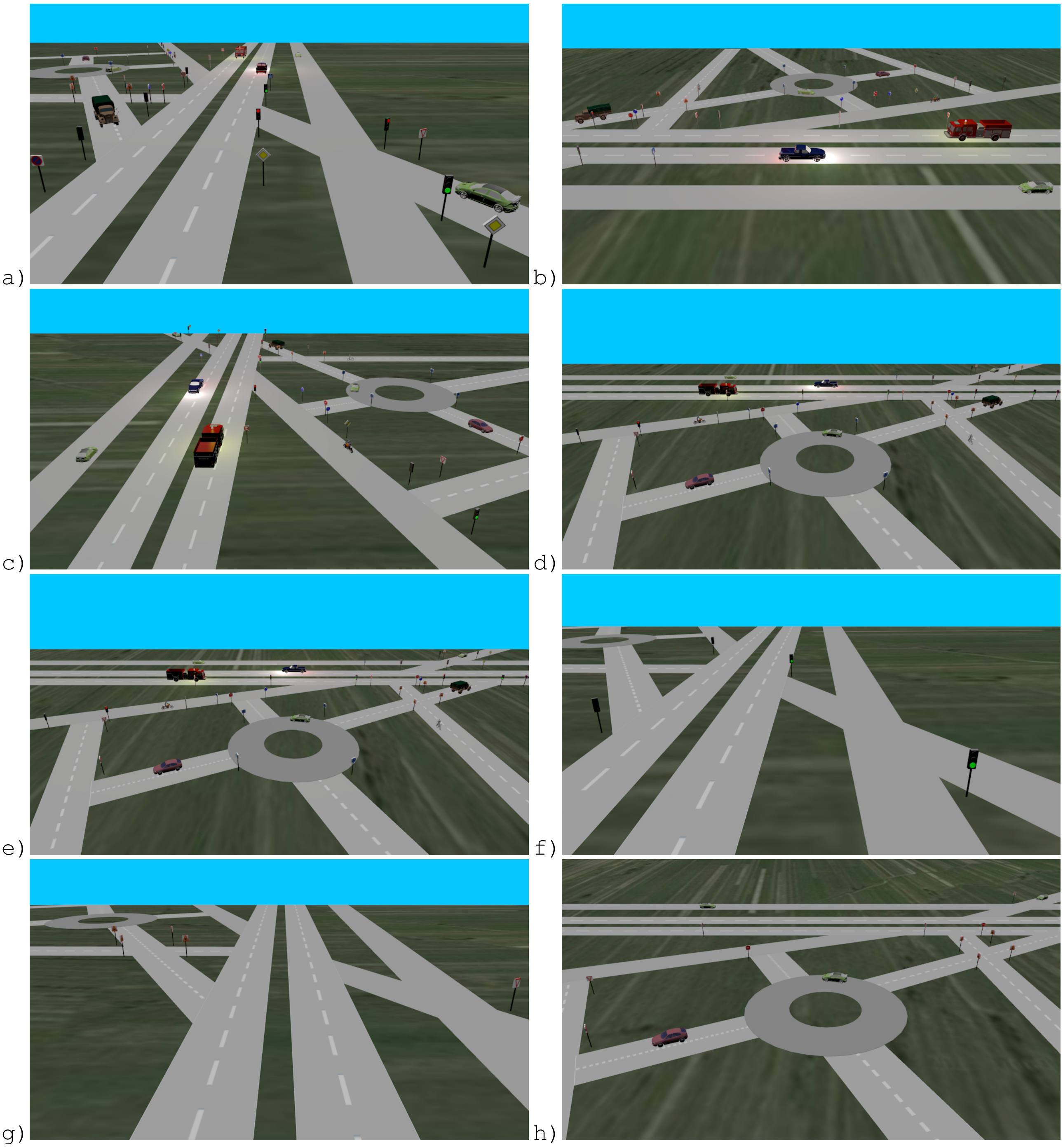


Figure 8.13: Crossroads with traffic lights and signs seen from different points (a-e), only green traffic lights are selected for presentation (f), only warning signs are selected for presentation (g), only warning and prohibition signs as well as passenger cars are selected for presentation (h)

which particular properties must be specified independently. Conceptual modeling with domain-specific properties also implies gain in mouse distance (2.68), as the user is liberated from performing manually complex operations, such as selecting different components of a car, and navigating across different controls of the environment to set individual coordinates of road signs. The average gain in the time required for modeling 3D scenes (1.74) is also caused by the difference in the number of properties that have to be changed in both environments.

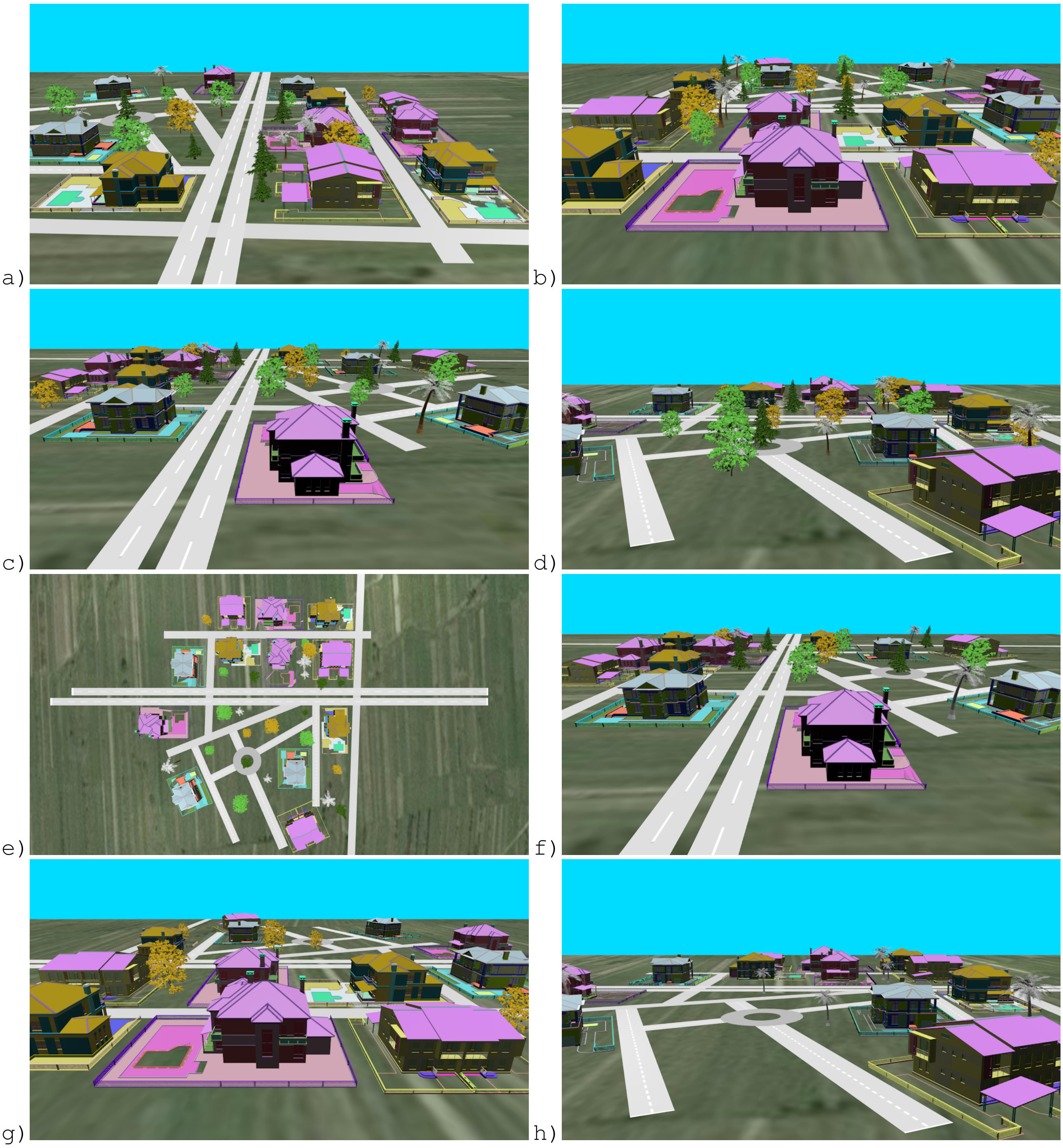


Figure 8.14: A housing with trees seen from different points, the trees differ in species, size and color of leafs (a-e), only big trees are selected for presentation (f), only trees with yellow leafs are selected for presentation (g), only palms are selected for presentation (h)

Customizing 3D Meta-Scenes

The created 3D scenes have been customized to satisfy different requirements. For every 3D scene created, the following customized scenes including only selected domain-specific objects have been generated.

1. *Scene 1*: Crossroads with cars, which have low and high beams (Figure 8.11a-d) customized into:
   - a scene including only cars with high beams (Figure 8.11e-h).

Metric	Scene 1		Scene 2		Scene 3		Scene 4		Average gain (B/S)
	B	S	B	S	B	S	B	S	
keystrokes and mouse clicks	864	224	674	170	1311	316	1326	333	3,99
mouse distance (inch)	2543	939	2235	951	4181	1653	4171	1340	2,68
time (s)	1390	854	1204	744	1721	1118	2504	1145	1,74

Table 8.4: Metrics calculated for the creation of 3D scenes (B—Blender, S—SO-SEMIC)

2. *Scene 2*: Crossroads with cars, some of which are priority vehicles (Figure 8.12a-d) customized into:
   - a scene including only priority vehicles (Figure 8.12e-h).
3. *Scene 3*: Crossroads with traffic lights, which are in different states (Figure 8.13a-e) customized into:
   - a scene including only green traffic lights (Figure 8.13f);
   - a scene including only warning signs (Figure 8.13g);
   - a scene including only warning and prohibition signs as well as passenger cars (Figure 8.13h).
4. *Scene 4*: A housing with trees, which have different scales and colors of leafs (Figure 8.14a-e) customized into:
   - a scene including only big trees (Figure 8.14f);
   - a scene including only trees with yellow leafs (Figure 8.14g);
   - a scene including only palms (Figure 8.14h).

The average values of the metrics (keystrokes and mouse clicks, mouse distance, and time) have been calculated for the customization of the 3D scenes (Table 8.5).

Metric	**Scene 1** only cars with high beam		**Scene 2** only priority vehicles		**Scene 3** warning signs		**Scene 3** green lights		**Scene 3** cars and signs		**Scene 4** big trees		**Scene 4** yellow trees		**Scene 4** palms		Average gain (B/S)
	B	S	B	S	B	S	B	S	B	S	B	S	B	S	B	S	
keystrokes and mouse clicks	110	5	68	5	15	5	21	5	12	4	6	4	8	4	7	5	6,34
mouse distance (inch)	237	10	138	12	53	12	73	12	36	10	42	11	29	12	22	10	7,22
time (s)	389	20	235	19	136	17	206	16	83	20	166	18	87	19	90	17	9,49

Table 8.5: Metrics calculated for the customization of 3D scenes (B—Blender, S—SO-SEMIC)

Unlike in the creation of 3D scenes, the number of keystrokes and mouse clicks (average gain 6.34) as well as mouse distance (average gain 7.22) in the customization of 3D scenes with SO-SEMIC do not depend on the complexity of the scenes being customized—since any customization requires only the selection of a single sub-class selector by a user. Complexity of the scenes affects the number of keystrokes and mouse clicks as well as mouse distance in content customization in Blender, because each object must be individually modified by a user. Complexity of scenes also affects the customization time (average gain 9.49) in both environments, as the more complex the scene, the more operations must be performed to customize it (cf. Section 8.2.4). The differences between the results obtained for the particular scenes are much bigger for Blender than for SO-SEMIC. In the customization of a scene, a user always performs a similar number of actions, which do not take much time in comparison to the actions performed by the customization algorithm. Hence, the average gain in time depends mostly on performance of the devices used, and it could be improved by using more efficient devices on both client and server sides. Average gain in the metrics increases with the increase of the number of objects to be modified in the scene. Thus, the highest gains have been obtained for Scene 1, from which multiple structural objects (cars) were removed, whereas the lowest gains have been obtained for Scene 4, from which less objects were removed. Therefore, automatic customization of 3D content is useful especially for complex scenes that require a large number of modifications, as it liberates users from completing the activities manually (navigating across different panels and selecting values of multiple properties).

Profit in the Size of Content Representations

Profit in the size of content representations from conceptual modeling is directly proportional to the number of modeled scenes and their size, and it is inversely proportional to the sum of the size of the RM (that must be implemented) and the size of the primary CpRs that are transformed to PSRs. The metric is given by the formula:

$$Profit = \frac{Size(PSRs)}{Size(RM) + Size(CpRs)} \tag{8.11}$$

The values of profit in the size of representations in relation to the number of scenes are presented in Figures 8.15-8.17. The creation of an RM is not profitable for low numbers of simple scenes (the values of profit lower than 1) and it is profitable for high numbers of complex scenes (the values grater than 1). The profit is higher for more verbose languages (ActionScript) and it is lower for more concise languages (VRML).

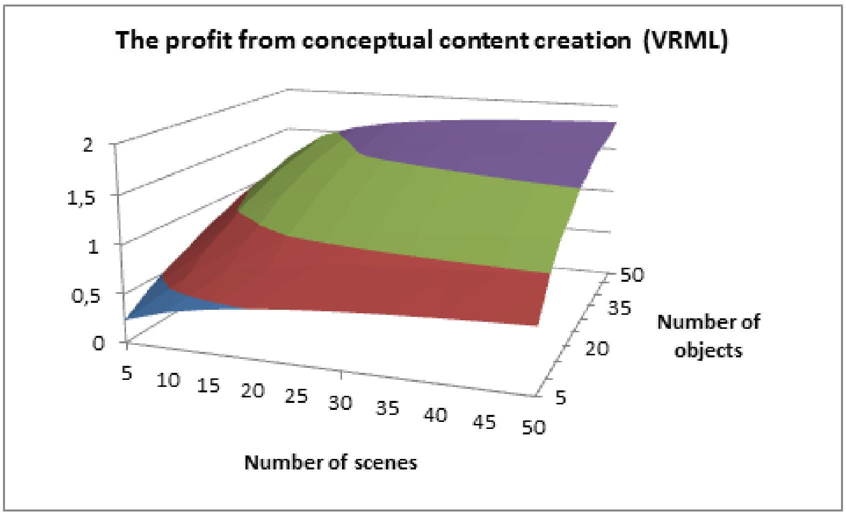


Figure 8.15: The profit from conceptual content creation for VRML (for representation size in bytes)

8.2.3. Computational Complexity of the Algorithms

The analysis of the computational complexity of the algorithms is based on the description of the stages of the algorithms presented in Sections 6.5.2, 6.6.3 and 6.7.2.

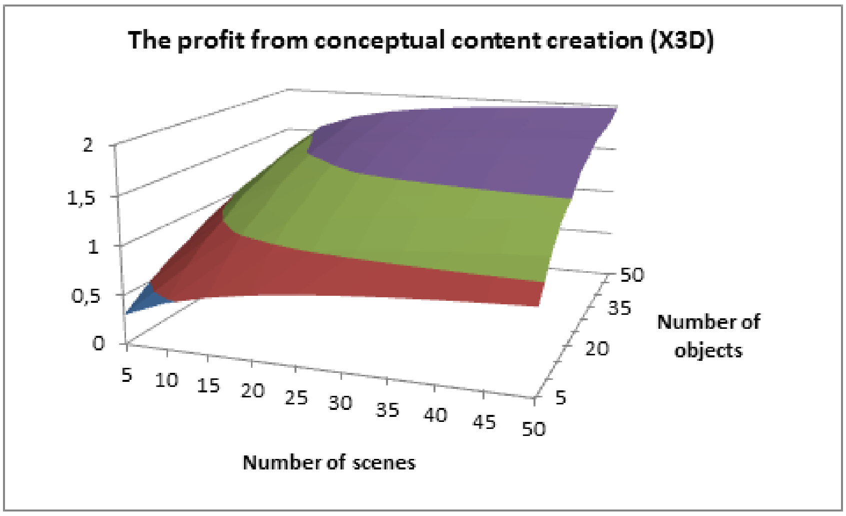


Figure 8.16: The profit from conceptual content creation for X3D (for representation size in bytes)

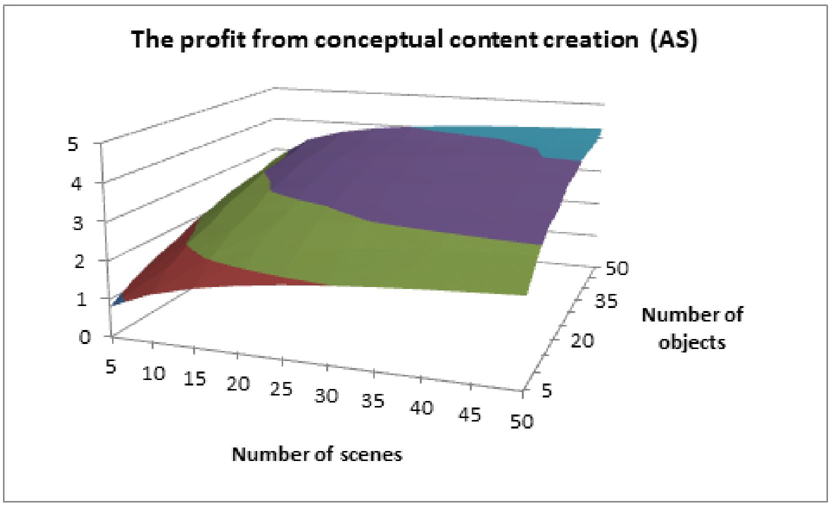


Figure 8.17: The profit from conceptual content creation for ActionScript (for representation size in bytes)

Computational Complexity of the Expanding Algorithm

In general, reasoning on RLs that are based on semantic rules, is undecidable, which implies that the expanding algorithm is undecidable. The computational complexity of the expanding algorithm may be determined, if no RLs are used in the processed SCR, and thereby, no undecidable reasoning is performed. For each step of the expanding algorithm, computational complexity has been determined. The following notation is used.

- n_P—the number of distinct domain-specific properties used in the SCR;
- n_{OP}—the number of distinct domain-specific object properties used in the SCR;
- n_S—the number of distinct statements in the SCR;
- $n_{S/OP}$—the number of distinct statements on domain-specific object properties in the SCR;
- n_{DI}—the number of distinct DIs in the SCR;

- n_{PO}—the number of distinct POs in the SCR;
- $n_{PO/CD}$—the average number of POs per CD in the SCR;
- n_{Cx}—the number of complex descriptors in the SCR;
- $n_{CD/Cx}$—the average number of CDs per complex descriptor in the SCR;
- $n_{MP/PO}$—the average number of distinct MPs per PO in the SCR;
- $n_{CD/PO}$—the average number of distinct CDs per PO in the SCR;
- n_c—the number of distinct cardinality restrictions in the SCR;
- n_{hv}—the number of distinct has-value restrictions in the SCR;
- N—the average cardinality of cardinality restrictions in the SCR.

1) In Step II-A, all the domain-specific properties used in the SCR are analyzed. Therefore, the computational complexity is $O(n_P)$.
2) In Step II-B, all the domain-specific object properties used in the SCR are analyzed. Therefore, the computational complexity is $O(n_{OP})$.
3) In Step II-C, all the statements with object properties in the SCR are analyzed. Therefore, the computational complexity is $O(n_{S/OP})$.
4) In Step II-D, all the DIs are analyzed. Therefore, the computational complexity is $O(n_{DI})$.
5) In Step II-E, for every PO, $n_{MP/PO}$ MPs are analyzed, on average. Therefore, the computational complexity is $O(n_{PO}*n_{MP/PO})$.
6) In Step II-F, for every complex descriptor CD, $n_{CD/Cx}$ CDs are analyzed, on average. For every CD, all of its POs are analyzed. Therefore, the computational complexity is $O(n_{Cx}*n_{CD/Cx}*n_{PO/CD})$.
7) In Step III-A, for every PO, n_c cardinality restrictions are analyzed and for each of them N SCM individuals are generated, on average. Therefore, the computational complexity is $O(n_{PO}*n_c*N)$.
8) In Step III-B, for every PO, n_{hv} has-value restrictions are analyzed. Therefore, the computational complexity is $O(n_{PO}*n_{hv})$.

For the aforementioned formulas, the upper bound of computational complexity may be specified. The number of distinct SCM classes or individuals is less or equal to $2*n_S$, since SCM classes and individuals may occur in statements as either the subject or the object. The number of distinct SCM properties is less or equal to n_S, since in every statement, only one SCM property occurs. Thus, overall the computational complexity of Stage II is polynomial with the upper bound $O(n_S{}^3)$. Overall computation complexity of Stage III is polynomial with the upper bound $O(n_S{}^2)$. To conclude, the expanding algorithm is undecidable when using RLs, and it has the polynomial computational complexity equal to $O(n_S{}^3)$, when using no RLs.

Computational Complexity of the Customization Algorithm

Like the expanding algorithm, the customization algorithm is undecidable unless RLs are not processed in Steps I-B-8, II-A and III-A-2. For each step of the customization algorithm, computational complexity has been determined. The following notation is used.

- n_I—the number of distinct individuals in the meta-scene;
- n_{IS}—the number of distinct individual selectors in the query;
- $n_{I/IS}$—the average number of distinct individuals per individual selector in the query;
- $n_{I/C}$—the average number of distinct individuals per SCM class in the meta-scene;
- n_P—the number of distinct SCM properties in the meta-scene;
- n_{SS}—the number of distinct sub-class selectors in the query;
- $n_{C/Sel}$—the average number of distinct SCM subclasses per selector in the query;

- n_{hv}—the number of distinct has-value selectors in the query;
- n_{avf}—the number of distinct all-values-from selectors in the query;
- n_{svf}—the number of distinct some-values-from selectors in the query;
- n_{IS}—the number of distinct intersection selectors in the query;
- $n_{Sel/IS}$—the average number of distinct selectors per intersection selector in the query;
- n_{DS}—the number of distinct difference selectors in the query;
- $n_{Sel/DS}$—the average number of distinct selectors per difference selector in the query;
- n_S—the number of distinct statements in the meta-scene;
- $n_{S/MS \cup Q}$—the number of distinct statements in the union of the meta-scene and the query;
- n_{SP}—the number of distinct sub-projectors in the query;
- $n_{P/SP}$—the average number of distinct subproperties per sub-projector in the query.

1) In Step I-B-1, for every individual selector, $n_{I/IS}$ individuals are analyzed, on average. Therefore, the computational complexity is $O(n_{IS}*n_{I/IS})$.
2) In Step I-B-2, for every sub-class selector, $n_{C/Sel}$ subclasses are analyzed, on average. For every subclass, $n_{I/C}$ individuals are analyzed, on average. Therefore, the computational complexity is $O(n_{SS}*n_{C/Sel}*n_{I/C})$.
3) In Step I-B-3, for every has-value selector, all individuals in the meta-scene are analyzed. Therefore, the computational complexity is $O(n_{hv}*n_I)$.
4) In Step I-B-4, for every all-values-from selector, all individuals in the meta-scene are analyzed. Therefore, the computational complexity is $O(n_{avf}*n_I)$.
5) In Step I-B-5, for every some-values-from selector, all individuals in the meta-scene are analyzed. Therefore, the computational complexity is $O(n_{svf}*n_I)$.
6) In Step I-B-6, for every intersection selector, $n_{Sel/IS}$ selectors are analyzed, on average. For every selector, $n_{C/Sel}$ SCM classes, each including $n_{I/C}$ individuals, are analyzed, on average. Therefore, the computational complexity is $O(n_{IS}*n_{Sel/IS}*n_{C/Sel}*n_{I/C})$.
7) In Step I-B-7, for every difference selector, $n_{Sel/DS}$ selectors are analyzed, on average. For every selector, $n_{C/Sel}$ SCM classes, each including $n_{I/C}$ individuals, are analyzed, on average. Therefore, the computational complexity is $O(n_{DS}*n_{Sel/DS}*n_{C/Sel}*n_{I/C})$.
8) In Step II-C, for every individual, all the statements in the mate-scene are analyzed. Therefore, the computational complexity is $O(n_I*n_S)$.
9) In Step III-A-1, for every sub-projector, $n_{P/SP}$ SCM properties are analyzed, on average. Therefore, the computational complexity is $O(n_{SP}*n_{P/SP})$.
10) In Step III-B, for every SCM property in the meta-scene, all the statements are analyzed. Therefore, the computational complexity is $O(n_P*n_S)$.

Every factor in the aforementioned formulas is less or equal to $n_{S/MS \cup Q}$. Hence, without using RLs, the overall computational complexity of content selection is $O({n_{S/MS \cup Q}}^4)$. The overall computational complexity of content modification is equal to complexity of the expanding algorithm (which is used within the customization algorithm), so it is $O({n_S}^3)$. The overall computational complexity of content projection is $O({n_{S/MS \cup Q}}^2)$. To conclude, the customization algorithm is undecidable when using RLs, and it has polynomial computational complexity with the upper bound equal to $O({n_{S/MS \cup Q}}^4)$—when using no RLs.

Computational Complexity of the Transformation Algorithm

For each stage of the transformation algorithm, computational complexity has been determined. The following notation is used.

- n_{SSPt}—the number of SSPts in the TKB,
- n_{SLPt}—the number of SLPts in the TKB,
- n_{SPt}—the number of SPts in the TKB,
- n_{SS}—the number of SSs in the TKB,
- $n_{S/PIR}$—the number of Ss in the PIR,
- $n_{S/TKB}$—the number of Ss in the TKB,
- $n_{S/SS}$—the average number of Ss per SS,
- $n_{SPt/SSPt}$—the average number of SPts per SSPt,
- $n_{TP/S}$—the average number of TPs per S,
- $n_{T/S}$—the average number of Ts per S.

1) At Stage I, every SPt occurring in an SSPt is processed once to be included in a SPARQL query. Therefore, the computational complexity is $O(n_{SSPt}*n_{SPt/SSPt})$.
2) At Stage II, creating SSs requires, in the worst case, for every SSPt (which determines a single SPARQL query), to check every SPt included in the SSPt against every S included in the PIR. Therefore, the computational complexity is $O(n_{SSPt}*n_{SPt/SSPt}*n_{S/PIR})$.
3) At Stage III, every TP must be set. Therefore, the computational complexity of this stage is polynomial and it is equal to $O(n_{S/PIR}*n_{TP/S})$.
4) At Stage IV, in the worst case, every T must be nested into another T by a TP. Therefore, the computational complexity of this stage is $O(n_{S/PIR}*n_{T/S})$.
5) At Stage V, creating a *global SLPt* requires, in the worst case, checking every SPt against every SLPt for all SPts included in the TKB. Therefore, the computational complexity is $O(n_{SLPt}*n_{SPt}^2)$.

Complexities of the consequent stages of the transformation algorithm are polynomial. The factors in the formulas are less or equal to the numbers of statements in the PIR and TKB. Thus, overall complexity of the transformation algorithm is polynomial with the upper bound $O(n_{S/PIR}^2+n_{S/TKB}^3)$.

8.2.4. Performance of the Algorithms

Performance of the algorithms used in SCCM has been evaluated. The tests have been performed for scenes whose number of objects changed over the range 5 to 50 with the step equal to 5. Every object in the scene is described by a position, orientation, scale and material. For every number of objects, 20 scenes have been generated and average results have been calculated.

Expanding Algorithm

Performance of the expanding algorithm (cf. Section 6.5) has been evaluated in terms of time required for creation of a PIR based on a CpR (Figure 8.18). Time required for expanding a CpR linearly increases with the increase of the number of objects. The tests cover creation of interconnected instances of concrete components for every domain-specific individual, on the basis of CDs (Section 8.2.3/Step II-E, Stage III). Since the average number of distinct MPs per PO as well as the number of restrictions in the SCR are constant, the computational complexity of expanding is linear $O(n_S)$, where S is the number of statements in the created meta-scene. For instance, if every tree of a class includes two sub-objects of some classes (leafs and branches), which have individual materials and relative positions (described by other classes), the creation of a particular tree covers the creation of the SCM individuals representing the sub-objects, their materials and relative positions, and assigning appropriate SCM data properties to these SCM individuals.

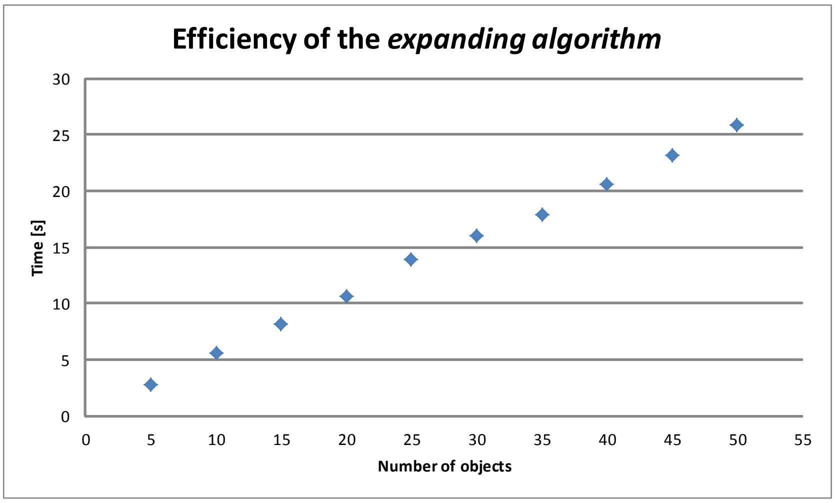


Figure 8.18: Time of expanding *conceptual 3D content representations*

Customization Algorithm

Performance of the customization algorithm (cf. Section 6.6) has been evaluated in terms of the time required for customization of a 3D meta-scene (Figure 8.19). The figure presents polynomial time required for customization of meta-scenes. The dependency follows from the time required for finding the selected SCM individuals and filtering out the statements that do not include either selected SCM individuals or selected SCM properties (Section 8.2.3/Steps I-B-2, II-C and Stage 3). Since the number of sub-class selectors, the average number of distinct SCM subclasses per selector and the number of distinct SCM individuals per SCM class are constant, the computational complexity of customization is $O(n_{S/MS \cup Q}{}^2)$. For instance, selecting the `dso:PassengerCar` class as the sub-class selector CCP entails finding all its SCM individuals with their required SCM properties indicating wheels, doors, lights, materials, etc.

Transformation Algorithm

Performance of the transformation algorithm (cf. Section 6.7) has been evaluated in terms of time required for the transformation of a PIR to a PSR (Figure 8.20). The graph presents polynomial time required for the transformation of PIRs, which are encoded with the CRO, to PSRs, which are encoded with the selected 3D content representation languages (X3D, VRML and ActionScript). Transformation time for VRML and X3D is similar, as the languages are supported by a common TKB and structurally equivalent TBs. Moreover, transformation time for VRML and X3D is more than twice lower than the transformation time for ActionScript. The difference is caused by the remarkably different structures of VRML/X3D and ActionScript representations. While transformation for VRML/X3D is based mainly on nesting Ts, which is relatively low time-consuming, transformation for ActionScript is based mainly on setting TPs, which is relatively high time-consuming, because the number of TPs is higher than the number of Ts—on average, more than two TPs are included in a T. Since the number of statements in the

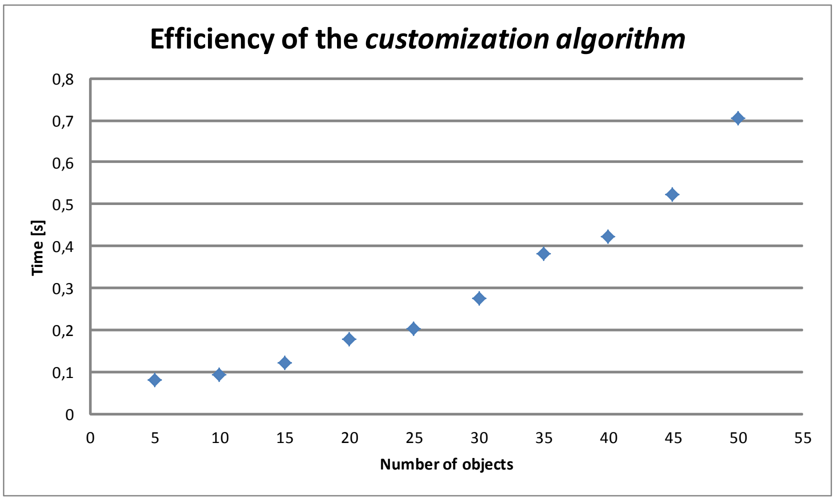


Figure 8.19: Time of customizing *3D meta-scenes*

TKBs is constant, the overall computational complexity of transformation is $O(n_{S/PIR}^{2})$—as explained in Section 8.2.3.

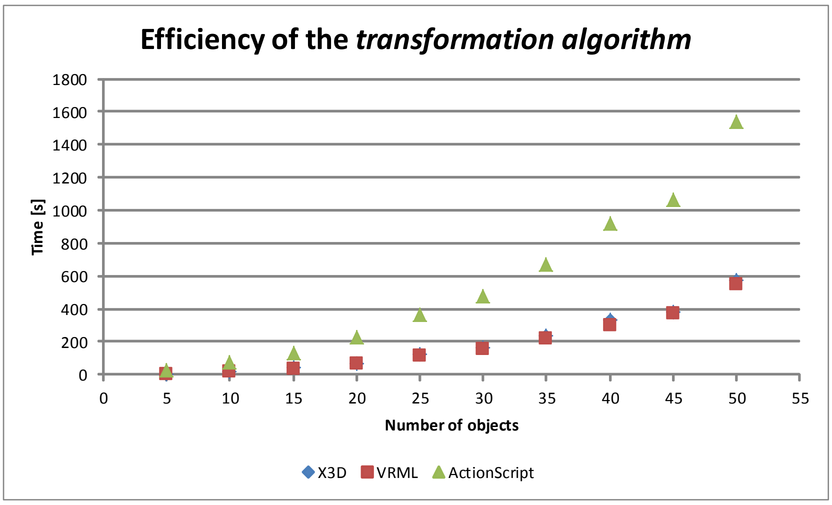


Figure 8.20: Time of transforming *Platform Independent Representations* to *Platform Specific Transformations*

8.3. Discussion

The tests carried out show that the SEMIC approach outperforms the previous approaches to modeling 3D content in terms of functionality, properties of produced content representations and user's effort in modeling content.

The qualitative tests show the lack of advanced functionality related to conceptual knowledge-based creation of multi-platform 3D content in the previous approaches. Functionality of the available solutions mostly focuses on the basic semantic constructions, which are processed within specific systems, and do not provide general methods that are capable of processing complex semantic statements to produce 3D content.

The advantage in terms of the size and complexity of content representations follows from modeling 3D content at a higher level of abstraction and leads to creating high-level objects that combine multiple low-level content components. Although the overall size and complexity of an RM and a CpR may be higher than the size and complexity of the resulting VRML, X3D or ActionScript scene, the RMs are created only once, they are common for all CpRs compliant with a particular DSO, and they are the only documents created by domain experts in the process of modeling with SEMIC. The size of CpRs is typically much smaller than the size of the corresponding PSRs, which have been encoded with the widely-used 3D content representation languages and programming libraries. The smaller size can enhance storage and transmission of 3D content to target users, e.g., from distributed web repositories to content presentation platforms. The values of size, length, volume, difficulty, effort and time calculated for PIRs are much higher than the values calculated for VRML/X3D representations. However, PIRs are generated automatically on the basis of CpRs, so it is not a burden on content authors to create them.

Furthermore, SEMIC provides a significant advantage in implementation of content (requires less effort and shorter time) by enabling the use of representation schemes whose complexity can be lower than complexity of frequently used 3D content representation languages. As a result, the vocabulary, length, volume and difficulty of content representations are lower and the creation as well as the understanding of the representations requires less effort and time. Hence, SEMIC can be used to accelerate and improve effectiveness of 3D content creation using domain-specific ontologies and tools that are less complicated than the tools currently used for 3D content creation with the available 3D content representation languages.

Moreover, SEMIC enables content creation by users without considerable expertise in 3D modeling, in particular experts in different domains. The calculated values of the profit from conceptual creation of 3D content show that the use of SEMIC is effective even for systems with relatively low numbers of scenes that have to be designed, in comparison to modeling all the scenes using the available languages.

In the context of multi-platform content representation, the results show high profit from the implementation and the use of a new transformation when adding a new platform in applications with a high number of scenes in contrast to the implementation of the counterparts of the scenes with a new 3D content representation language supported by the new platform. The profit is also high when PSRs are encoded using programming libraries that do not provide some required components and properties, which need to be implemented from scratch. In such cases, the high profit is the result of high re-usability of such components and SCM properties which, once implemented in Ts, may be reused multiple times.

The results show that the user's effort in modeling of 3D content with SEMIC is much lower than the effort in modeling of 3D content with other environments and languages. The advantage includes both the modeling actions accomplished by the user (the number of keystrokes and mouse clicks as well as mouse distance) as well as time required for modeling. The predominance follows from modeling 3D content at a conceptual level, with domain-specific classes and properties, which liberates users from direct specification of details related to computer graphics. Hence, the modeling actions become less complex and time-consuming—do not demand changing multiple properties attainable on different panels of the modeling environment.

The tests carried out indicate relatively long time required for processing semantic 3D content representations, in particular expanding and transforming them. However, performance of the algorithms could be improved, e.g., by using multi-threading, by employing a more efficient SPARQL query processing engine and more efficient devices.

9. Conclusions

Despite significant research effort in this domain and the development of numerous solutions facilitating 3D content creation, creation of interactive 3D content is still a challenging task, as it requires advanced technical skills in modeling various elements of 3D content, such as geometry, structure, appearance, animation and behavior. The available solutions for modeling 3D content encompass: imperative and declarative programming languages, visual modeling environments as well as approaches to 3D content configuration based on reusable content components. Applying semantic techniques to modeling of interactive 3D content receives increasing attention from the research community, as it provides new opportunities that go beyond the current state of the art:

1) declarative modeling of 3D content based on knowledge discovery,
2) flexible conceptual representation of 3D content at different levels of abstraction,
3) separation of concerns in 3D content creation between different modeling users,
4) on-demand 3D content customization by different content consumers (users and applications),
5) multi-platform 3D content presentation.

The SEMIC approach to semantic modeling of interactive 3D content proposed in this dissertation, leverages the semantic web standards to enable modeling of 3D content for modern VR/AR systems and to satisfy the requirements specified in Section 4.1. SEMIC goes beyond the current state of the art in semantic modeling of 3D content in several respects:

1. SEMIC enables *declarative content creation*, which is focused on the description of the results to be achieved, instead of the manner in which the results are to be obtained.
2. SEMIC enables *conceptual content creation* at different levels of abstraction by the use of different application- or domain-specific ontologies.
3. SEMIC permits *inference of tacit knowledge*, which is not explicitly specified by content authors, but which can be discovered and which affects the results of modeling.
4. SEMIC makes possible *separation of concerns* in modeling content between different users with different skills, who are equipped with different modeling tools.
5. SEMIC enables *on-demand content customization* by different content consumers based on generalized 3D content representations.
6. SEMIC enables flexible *multi-platform content presentation*.

The main research achievements of this dissertation are the following:

1. The *classification* of the available approaches to semantic representation and modeling of interactive 3D content (Chapters 2 and 3). The classification is based on various criteria and covers a large number of approaches. The classification and review of the available approaches has enabled comprehensive discussion of the current state of the art in the field of semantic representation and modeling of 3D content. It has also permitted to specify requirements for modeling 3D content for modern 3D/VR/AR applications.

2. *Declarative, conceptual representation* of 3D content based on SCM (Chapter 5). The representation leverages widely accepted semantic web standards as well as application- or domain-specific ontologies. It has been shown that the conceptual content representation has lower size and complexity than content representations encoded in existing 3D content representation languages (Chapter 8).
3. *Multi-platform representation* of 3D content based on SCM (Chapter 5). The representation is platform- and standard-independent and enables flexible generation of 3D content encoded in different formats and languages. It has been shown that the transformation of multi-platform content representations to representations encoded in selected languages has higher efficiency than the modeling of individual representations in these content representation languages (Chapter 8).
4. *Ontologies* that enable semantic 3D content representation at different levels of abstraction and are used in semantic transformation of 3D content to different content representation languages (Chapter 5).
5. Semantic *knowledge-based creation* of 3D content with SCCM with the use of domain-specific ontologies, which enable separation of concerns between different modeling users (Chapter 6). It has been shown that the semantic creation of 3D content is more efficient in terms of the required time and the number of operations than the creation performed using representative 3D modeling environments (Chapter 8).
6. Semantic *on-demand customization* of 3D content with SCCM, which is based on domain-specific ontologies (Chapter 6). It has been shown that the semantic customization of 3D content is more efficient in terms of the required time and the number of operations than the customization performed using representative 3D modeling environments (Chapter 8).
7. The *SO-SEMIC environment*, which enables efficient modeling of interactive 3D content (Chapter 7).

To summarize, the goal of this dissertation has been achieved. The evaluation presented in Chapter 8 proves that *the SEMIC approach enables efficient creation of interactive 3D content at the conceptual level, using domain-specific ontologies*.

Possible directions of future research and development activities include several aspects.

1. Persistent link between semantic and final (non-semantic) 3D content representations can be proposed to enable real-time synchronization of content representations. Such synchronization can lead to the development of dynamically explorable and manipulable 4-dimensional 3D content including space and time. Dynamic explorable and manipulable VR/AR scenarios could be accessed by different users querying and changing the content through semantic interfaces, and observing the results in real-time.
2. Transformation of declarative rule-based descriptions of complex content behavior to different 3D content languages can be elaborated. Such transformation should be performed in a different manner than the transformation of statements, as rules describing content behavior cannot be used for inference before being transformed. Instead, the rules must be transformed to appropriate (e.g., conditional) instructions of the target language to maintain premises and consequences, which is necessary to preserve the dynamism of content behavior.
3. Semantic representation of the context of interaction can be added to SEMIC to enable seamless adaptation of 3D content with regards to user location, preferences, parameters of the hardware and software platforms used, etc.

4. Distributed semantic 3D content representations can be proposed as an extension of the SCM-based representations with possibly stateful 3D content components maintained in different hosts and dynamically composed into 3D scenes over the network.

This research work has been supported by the Polish National Science Centre (NCN) Grants No. DEC-2012/07/B/ST6/01523 and DEC-2014/12/T/ST6/00039.

Abbreviations

2D	—	Two-dimensional
3D	—	Three-dimensional
AR	—	Augmented Reality
BIM	—	Building Information Models
CCP	—	3D Content Customization Pattern (SEMIC)
CF	—	Code Fragment (SEMIC)
CO	—	Customization Ontology (SEMIC)
CpR	—	Conceptual 3D Content Representation (SEMIC)
CRO	—	Concrete 3D Content Ontology (SEMIC)
CrR	—	Concrete 3D Content Representation (SEMIC)
DC	—	Descriptive Class (SEMIC)
DI	—	Descriptive Individual (SEMIC)
DIC	—	Descriptive Individual Class (SEMIC)
DSO	—	Domain-specific Ontology (SEMIC)
DXF	—	Data Exchange Format
IFC	—	Industry Foundation Classes
KRL	—	Knowledge Representation Layer
LLOC	—	Logical Lines of Code
MDP	—	Mapping Data Property (SEMIC)
MO	—	Mapping Ontology (SEMIC)
MOP	—	Mapping Object Property (SEMIC)
MP	—	Mapping Property (SEMIC)
MPEG-7	—	Multimedia Content Description Interface
MVC	—	Model-View-Controller
OWL	—	Web Ontology Language
PIR	—	Platform-independent 3D Content Representation (SEMIC)
PO	—	Presentable Object (SEMIC)
POC	—	Presentable Objects Class (SEMIC)
PSR	—	Platform-specific 3D Content Representation (SEMIC)
RAIVE	—	REALISM Artificial Intelligence Virtual Environment
RDF	—	Resource Description Framework
RDFS	—	RDF Schema
RL	—	Relation (SEMIC)
RM	—	Representation Mapping (SEMIC)
S	—	Statement (SEMIC)
SC	—	Statement Collection (SEMIC)
SCCM	—	Semantic 3D Content Creation Method (SEMIC)
SCM	—	Semantic 3D Content Model (SEMIC)

SCPt	—	Statement Collection Pattern (SEMIC)
SCR	—	Semantic 3D Content Representation (SEMIC)
SEMIC	—	Semantic Modeling of Interactive 3D Content
SL	—	Statement List (SEMIC)
SLPt	—	Statement List Pattern (SEMIC)
SNIL	—	Semantic Net Interchange Language
SOA	—	Service-Oriented Architecture
SO-SEMIC	—	Service-Oriented Semantic Modeling of Interactive 3D Content (SEMIC)
SPt	—	Statement Pattern (SEMIC)
SS	—	Statement Set (SEMIC)
SSPt	—	Statement Set Pattern (SEMIC)
SWRL	—	Semantic Web Rule Language
T	—	Template (SEMIC)
TB	—	Template Base (SEMIC)
TKB	—	Transformation Knowledge Base (SEMIC)
TO	—	Transformation Ontology (SEMIC)
TP	—	Template Parameter (SEMIC)
TPPt	—	Template Parameter Pattern (SEMIC)
TPPtS	—	Template Parameter Pattern Set (SEMIC)
TPt	—	Template Pattern (SEMIC)
TS	—	Template Set (SEMIC)
TSPt	—	Template Set Pattern (SEMIC)
UML	—	Unified Modeling Language
VDE	—	Virtual Design Environment
VR	—	Virtual Reality
VRML	—	Virtual Reality Modeling Language
W3C	—	World-Wide Web Consortium
X3D	—	Extensible 3D

Bibliography

[1] Autocad civil 3d. http://www.autodesk.com/products/autocad-civil-3d/overview, accessed March 20, 2015.

[2] Away3d. http://away3d.com/, accessed March 20, 2015.

[3] Direct3d 11.1 features. https://msdn.microsoft.com/en-us/library/windows/desktop/hh404562%28v=vs.85%29.aspx, accessed March 20, 2015.

[4] Freewrl home page. http://freewrl.sourceforge.net/, accessed March 20, 2015.

[5] Ghost productions: Medical animation, illustration & interactive media. http://www.ghostproductions.com/, accessed March 20, 2015.

[6] Instant reality. http://www.instantreality.org/, accessed March 20, 2015.

[7] Javascript for acrobat 3d annotations api reference. http://www.adobe.com/content/dam/Adobe/en/devnet/acrobat/pdfs/js_3d_api_reference.pdf, accessed March 20, 2015.

[8] Jogl. http://jogamp.org/jogl/www/, accessed March 20, 2015.

[9] Opengl. https://www.opengl.org/, accessed March 20, 2015.

[10] Sandy: 3d engine for flash and haxe platforms. http://code.google.com/p/sandy/, accessed March 20, 2015.

[11] Sweet home 3d. http://www.sweethome3d.com/, accessed March 20, 2015.

[12] Unity. http://unity3d.com/5, accessed March 20, 2015.

[13] Universal 3d file format. http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-363%204th%20Edition.pdf, accessed March 20, 2015.

[14] Unreal engine. https://www.unrealengine.com/what-is-unreal-engine-4, accessed March 20, 2015.

[15] X3dom. http://www.x3dom.org/, accessed March 20, 2015.

[16] Xml3d. http://xml3d.org/, accessed March 20, 2015.

[17] 3d-studio file format. http://www.martinreddy.net/gfx/3d/3DS.spec, accessed March 24, 2015.

[18] 3dsa: Semantic annotations for 3d artefacts. http://www.itee.uq.edu.au/eresearch/projects/3dsa, accessed March 24, 2015.

[19] 3dvia. http://www.3dvia.com/, accessed March 24, 2015.

[20] Aboutwordnet. http://wordnet.princeton.edu, accessed March 24, 2015.

[21] Aim@shape project. http://cordis.europa.eu/ist/kct/aimatshape_synopsis.htm, accessed March 24, 2015.

[22] Alternativaplatform. http://old.alternativaplatform.com/en/technologies/alternativa3d/, accessed March 24, 2015.

[23] Autocad civil 3d. http://www.autodesk.com/products/autocad-civil-3d/overview, accessed March 24, 2015.

[24] Bitmanagement bs contact. http://www.bitmanagement.com/products/interactive-3d-clients/bs-contact, accessed March 24, 2015.

[25] Blender. http://www.blender.org/, accessed March 24, 2015.

[26] Blender documentation. http://www.blender.org/api/blender_python_api_2_73_release/, accessed March 24, 2015.

[27] Blends / base low-poly man. http://www.blendswap.com/blends/view/2949, accessed March 24, 2015.

[28] Blends / bike. http://www.blendswap.com/blends/view/58595, accessed March 24, 2015.

[29] Blends / car 739. http://www.blendswap.com/blends/view/56683, accessed March 24, 2015.

[30] Blends / cottages pack. http://www.blendswap.com/blends/view/69406, accessed March 24, 2015.

[31] Blends / firetruck. http://www.blendswap.com/blends/view/69906, accessed March 24, 2015.

[32] Blends / fixed gear bike. http://www.blendswap.com/blends/view/68326, accessed March 24, 2015.

[33] Blends / house test. http://www.blendswap.com/blends/view/68817, accessed March 24, 2015.

[34] Blends / mitsubishi lancer x. http://www.blendswap.com/blends/view/71006, accessed March 24, 2015.

[35] Blends / opel blitz kfz 305. http://www.blendswap.com/blends/view/66103, accessed March 24, 2015.

[36] Blends / palm tree cycles ready. http://www.blendswap.com/blends/view/64383, accessed March 24, 2015.

[37] Blends / pine tree. http://www.blendswap.com/blends/view/63269, accessed March 24, 2015.

[38] Blends / tree sketch tree #1. http://www.blendswap.com/blends/view/61605, accessed March 24, 2015.

[39] Blends / uk road signs - pack 1 updated version. http://www.blendswap.com/blends/view/67283, accessed March 24, 2015.

[40] Citygml. http://www.opengeospatial.org/standards/citygml, accessed March 24, 2015.

[41] A class-based verb lexicon. http://verbs.colorado.edu/, accessed March 24, 2015.

[42] Collada - digital asset and fx exchange schema. https://collada.org/mediawiki/index.php/Extension, accessed March 24, 2015.

[43] Collada - digital asset schema release 1.4.1, specification (2nd edition). https://www.khronos.org/files/collada_spec_1_4.pdf, accessed March 24, 2015.

[44] Cortona 3d. http://www.cortona3d.com/cortona3d-viewers, accessed March 24, 2015.

[45] Daml+oil (march 2001) reference description. http://www.w3.org/TR/daml+oil-reference, accessed March 24, 2015.

[46] Extensible 3d (x3d), part 1: Architecture and base components, core component. http://www.web3d.org/documents/specifications/19775-1/V3.2/Part01/components/core.html\#Nodereference, accessed March 24, 2015.

[47] Fbx file structure. http://wiki.blender.org/index.php/User:Mont29/Foundation/FBX_File_Structure, accessed March 24, 2015.

[48] Ghost productions. http://www.ghostproductions.com/, accessed March 24, 2015.

[49] Glge: Webgl for the lazy. http://www.glge.org/, accessed March 24, 2015.

[50] Industry foundation classes release 4 (ifc4). http://www.buildingsmart-tech.org/ifc/IFC4/final/html/index.htm, accessed March 24, 2015.

[51] Jebgl: Java emulated webgl canvas. http://code.google.com/p/jebgl/, accessed March 24, 2015.

[52] json - json encoder and decoder. https://docs.python.org/2/library/json.html, accessed March 24, 2015.

[53] Microsoft virtual academy, introduction to asp.net mvc. http://www.microsoftvirtualacademy.com/training-courses/introduction-to-asp-net-mvc, accessed March 24, 2015.

[54] Obj specification. http://www.martinreddy.net/gfx/3d/OBJ.spec, accessed March 24, 2015.

[55] Object constraint language (ocl). http://www.omg.org/spec/OCL/, accessed March 24, 2015.

[56] Owl 2 web ontology language structural specification and functional-style syntax. http://www.w3.org/TR/owl2-syntax/, accessed March 24, 2015.

[57] Owl web ontology language reference. http://www.w3.org/TR/owl-ref/, accessed March 24, 2015.

[58] Papervision3d: Open source realtime 3d engine for flash. http://code.google.com/p/papervision3d/, accessed March 24, 2015.

[59] Prism - partnership for research in spatial modeling. http://prism.engineering.asu.edu/research/3dk.php, accessed March 24, 2015.

[60] Protégé. http://protege.stanford.edu/, accessed March 24, 2015.

[61] Rdf 1.1 turtle. http://www.w3.org/TR/turtle/, accessed March 24, 2015.

[62] Rdf schema 1.1. http://www.w3.org/TR/rdf-schema/, accessed March 24, 2015.

[63] Restlet api platform. http://restlet.com/, accessed March 24, 2015.

[64] Sketchup. http://www.sketchup.com/, accessed March 24, 2015.

[65] Sweet home 3d. http://www.sweethome3d.com/, accessed March 24, 2015.

[66] Techtarget. http://searchsoftwarequality.techtarget.com/definition/pattern, accessed March 24, 2015.

[67] Techtarget. http://searchsoa.techtarget.com/definition/service-oriented-architecture, accessed March 24, 2015.

[68] urllib — open arbitrary resources by url. https://docs.python.org/2/library/urllib.html, accessed March 24, 2015.

[69] Webgl. https://get.webgl.org/, accessed March 24, 2015.

[70] Webgl implementation of o3d. http://code.google.com/p/o3d/, accessed March 24, 2015.

[71] Xml schema. http://www.w3.org/XML/Schema.html, accessed March 24, 2015.

[72] Aivosto. Lines of code metrics (loc). http://www.aivosto.com/project/help/pm-loc.html, accessed March 24, 2015.

[73] R. Albertoni, L. Papaleo, M. Pitikakis, F. Robbiano, M. Spagnuolo, and G. Vasilakis. Ontology-based searching framework for digital shapes. In *On the Move to Meaningful Internet Systems 2005: OTM 2005 Workshops*, pages 896–905. Springer, 2005.

[74] Sven Albrecht, Thomas Wiemann, Martin Günther, and Joachim Hertzberg. Matching cad object models in semantic mapping. In *Proceedings ICRA 2011 Workshop: Semantic Perception, Mapping and Exploration, SPME*, 2011.

[75] Alexander Almer, Thomas Schnabel, Harald Stelzl, Jörg Stieg, and Patrick Luley. A tourism information system for rural areas based on a multi platform concept. In *Proceedings of the 6th International Conference on Web and Wireless Geographical Information Systems*, pages 31–41, Springer-Verlag Berlin Heidelberg, October 25-29, 2006.

[76] Tansu Alpcan, Christian Bauckhage, and Evangelos Kotsovinos. Towards 3d internet: Why, what, and how? In *Cyberworlds, 2007. CW'07. International Conference on*, pages 95–99. IEEE, 2007.

[77] Abhishek Anand, Hema Swetha Koppula, Thorsten Joachims, and Ashutosh Saxena. Contextually guided semantic labeling and search for three-dimensional point clouds. *Int. J. Rob. Res.*, 32(1):19–34, January 2013.

[78] Apache. Apache jena. http://jena.apache.org/, accessed March 24, 2015.

[79] ARCO. *Virtual Museum System*, accessed March 24, 2015.

[80] Marco Attene, Francesco Robbiano, Michela Spagnuolo, and Bianca Falcidieno. Semantic annotation of 3d surface meshes based on feature characterization. In *Proceedings of the Semantic and Digital Media Technologies 2Nd International Conference on Semantic Multimedia*, SAMT'07, pages 126–139, Berlin, Heidelberg, 2007. Springer-Verlag.

[81] Marco Attene, Francesco Robbiano, Michela Spagnuolo, and Bianca Falcidieno. Characterization of 3d shape parts for semantic annotation. *Comput. Aided Des.*, 41(10):756–763, October 2009.

[82] Autodesk. 3ds max. http://www.autodesk.pl/products/autodesk-3ds-max/overview, accessed March 24, 2015.

[83] Ruth Aylett and Marc Cavazza. Intelligent virtual environments: a state-of-the-art report. In *In: Eurographics 2001, STAR Reports volume*, pages 87–109, 2001.

[84] J. Benner, A. Geiger, and K. Leinemann. Flexible generation of semantic 3d building models. In *Proceedings of the 1st international workshop on next generation 3D city models, Bonn*, pages 17–22, 2005.

[85] Tim Berners-Lee, James Hendler, Ora Lassila, et al. The semantic web. *Scientific american*, 284(5):28–37, 2001.

[86] Ioan Marius Bilasco, J. Gensel, M. Villanova-Oliver, and H. Martin. On indexing of 3d scenes using mpeg-7. In *Proceedings of the 13th annual ACM international conference on Multimedia*, pages 471–474, Singapore, November 06-12, 2005.

[87] Ioan Marius Bilasco, Jérôme Gensel, Marlène Villanova-Oliver, and Hervé Martin. An mpeg-7 framework enhancing the reuse of 3d models. In *Proceedings of the Eleventh International Conference on 3D Web Technology*, Web3D '06, pages 65–74, New York, NY, USA, 2006. ACM.

[88] Ioan Marius Bilasco, Jérôme Gensel, Marlene Villanova-Oliver, and Hervé Martin. 3dseam: a model for annotating 3d scenes using mpeg-7. In *ISM*, pages 310–319. IEEE Computer Society, 2005.

[89] Ioan Marius Bilasco, Marlène Villanova-Oliver, Jérôme Gensel, and Hervé Martin. Semantic-based rules for 3d scene adaptation. In *Proceedings of the Twelfth International Conference on 3D Web Technology*, Web3D '07, pages 97–100, New York, NY, USA, 2007. ACM.

[90] Wesley Bille. *Conceptual Modeling of Complex Objects for Virtual Environments*. PhD thesis, Vrije Universiteit Brussel, 2006-2007.

[91] Wesley Bille, Olga De Troyer, Frederic Kleinermann, Bram Pellens, and Raul Romero. Using ontologies to build virtual worlds for the web. In Pedro T. Isaías, Nitya Karmakar, Luís Rodrigues, and Patrícia Barbosa, editors, *ICWI*, pages 683–690. IADIS, 2004.

[92] Wesley Bille, Bram Pellens, Frederic Kleinermann, and Olga De Troyer. Intelligent modelling of virtual worlds using domain ontologies. In *Proceedings of the Workshop of Intelligent Computing (WIC), held in conjunction with the MICAI 2004 conference*, pages 272–279, Mexico City, Mexico, 2004.

[93] Wesley Bille, Olga De Troyer, Bram Pellens, and Frederic Kleinermann. Conceptual modeling of articulated bodies in virtual environments. In H. Thwaites, editor, *Proceedings of the 11th International Conference on Virtual Systems and Multimedia (VSMM)*, pages 17–26, Ghent, Belgium, 2005. Archaeolingua, Archaeolingua.

[94] A. Borrmann. From gis to bim and back again – a spatial query language for 3d building models and 3d city models. In *Proceedings of the International ISPRS Conference on 3D Geoinformation*, Berlin, Germany, 2010.

[95] A. Borrmann and E. Rank. Query support for bims using semantic and spatial conditions. In J. Underwood and U. Isikdag, editors, *Handbook of Research on Building Information Modeling and Construction Informatics: Concepts and Technologies*. IGI Global, 2009.

[96] A. Borrmann and E. Rank. Specification and implementation of directional operators in a 3d spatial query language for building information models. *Advanced Engineering Informatics*, 23(1):32–44, 2009.

[97] André Borrmann and Ernst Rank. Topological analysis of 3d building models using a spatial query language. *Adv. Eng. Inform.*, 23(4):370–385, October 2009.

[98] Cédric Buche, Ronan Querrec, Pierre De Loor, and Pierre Chevaillier. Mascaret: Pedagogical multi-agents system for virtual environment for training. In *CW*, pages 423–431. IEEE Computer Society, 2003.

[99] Cédric Buche, Cyril Bossard, Ronan Querrec, and Pierre Chevaillier. Pegase: A generic and adaptable intelligent system for virtual reality learning environments. *International Journal of Virtual Reality*, 9(2):73–85, 2010.

[100] Marc Cavazza and Ian Palmer. High-level interpretation in virtual environments. *Applied Artificial Intelligence*, 14(1):125–144, 2000.

[101] Sasko Celakovski and Danco Davcev. Multiplatform real-time rendering of mpeg-4 3d scenes with microsoft xna. In *ICT Innovations 2009*, pages 337–344, Springer-Verlag Berlin Heidelberg, October 25-29, 2010.

[102] Siddhartha Chaudhuri, Evangelos Kalogerakis, Stephen Giguere, and Thomas Funkhouser. Attribit: Content creation with semantic attributes. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology*, UIST '13, pages 193–202, New York, NY, USA, 2013. ACM.

[103] Siddhartha Chaudhuri, Evangelos Kalogerakis, Leonidas Guibas, and Vladlen Koltun. Probabilistic reasoning for assembly-based 3d modeling. *ACM Trans. Graph.*, 30(4):35:1–35:10, July 2011.

[104] Chaomei Chen, Linda Thomas, Janet Cole, and Chiladda Chennawasin. Representing the semantics of virtual spaces. *IEEE MultiMedia*, 6(2):54–63, 1999.

[105] Pierre Chevaillier, ThanhHai Trinh, Mukesh Barange, Pierre De Loor, Frédéric Devillers, Julien Soler, and Ronan Querrec. Semantic modeling of virtual environments using mascaret. In *SEARIS*, pages 1–8. IEEE, 2012.

[106] Jacek Chmielewski. *Metadata Schema of Interactions for Multimedia Objects*. PhD thesis, Technical University of Gdańsk, 2007.

[107] Jacek Chmielewski. Interaction descriptor for 3d objects. In *Human System Interactions, 2008 Conference on*, pages 18–23. IEEE, 2008.

[108] Jacek Chmielewski. Interaction interfaces for unrestricted multimedia interaction descriptions. In *Proceedings of the 6th International Conference on Advances in Mobile Computing and Multimedia*, MoMM '08, pages 397–400, New York, NY, USA, 2008. ACM.

[109] Jacek Chmielewski. Metadata model for interactions of 3d objects. In *Information Technology, 2008. IT 2008. 1st International Conference on*, pages 1–4. IEEE, 2008.

[110] Jacek Chmielewski. Describing interactivity of 3d content. In Wojciech Cellary and Krzysztof Walczak, editors, *Interactive 3D Multimedia Content*, pages 195–221. Springer, 2012.

[111] Jacek Chmielewski. Finding interactive 3d objects by their interaction properties. *Multimedia Tools and Applications*, pages 1–26, 2012.

[112] Yu Lin Chu and Tsai Yen Li. Realizing semantic virtual environments with ontology and pluggable procedures. *Applications of Virtual Reality*, 2012.

[113] YuLin Chu and TsaiYen Li. 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*, VRCAI '08, pages 27:1–27:6, New York, NY, USA, 2008. ACM.

[114] Karin Coninx, Olga De Troyer, Chris Raymaekers, and Frederic Kleinermann. Vr-demo: a tool-supported approach facilitating flexible development of virtual environments using conceptual modelling. In *Proceedings of Virtual Concept*. Springer-Verlag, 2006.

[115] Raimund Dachselt, Michael Hinz, and Stefan Pietschmann. Using the amacont architecture for flexible adaptation of 3d web applications. In *Proceedings of the Eleventh International Conference on 3D Web Technology*, Web3D '06, pages 75–84, New York, NY, USA, 2006. ACM.

[116] Petros Daras, Apostolos Axenopoulos, Vasilios Darlagiannis, Dimitrios Tzovaras, Xavier Le Bourdon, Laurent Joyeux, Anne Verroust-Blondet, Vincenzo Croce, Thomas Steiner, Alberto Massari, Antonio Camurri, Steeve Morin, AmarDjalil Mezaour, Lorenzo Sutton, and Sabine Spiller. Introducing a unified framework for content object description. *IJMIS*, 2(3/4):351–375, 2011.

[117] S. Daum and A. Borrmann. Checking spatio-semantic consistency of building information models by means of a query language. In *Proceedings of the International Conference on Construction Applications of Virtual Reality*, 2013.

[118] S. Daum and A. Borrmann. Definition and implementation of temporal operators for a 4d query language. In *Proceedings of the ASCE International Workshop on Computing in Civil Engineering*. ASCE, 2013.

[119] L. De Floriani, A. Hui, L. Papaleo, M. Huang, and J. Hendler. A semantic web environment for digital shapes understanding. In *Semantic Multimedia*, pages 226–239. Springer, 2007.

[120] Leila De Floriani and Michela Spagnuolo. *Shape analysis and structuring*. Springer, 2007.

[121] Livio De Luca, Philippe Véron, and Michel Florenzano. A generic formalism for the semantic modeling and representation of architectural elements. *The Visual Computer*, 23(3):181–205, 2007.

[122] D. C. De Paiva, R. Vieira, and S. R. Musse. Ontology-based crowd simulation for normal life situations. In *Proceedings of the Computer Graphics International 2005*, CGI '05, pages 221–226, Washington, DC, USA, 2005. IEEE Computer Society.

[123] Olga De Troyer, Wesley Bille, Raul Romero, and Peter Stuer. On generating virtual worlds from domain ontologies. In *Proceedings of the 9th International Conference on Multi-Media Modeling*, pages 279–294, Taipei, Taiwan, 2003.

[124] Olga De Troyer, Frederic Kleinermann, Haïthem Mansouri, Bram Pellens, Wesley Bille, and Vladimir Fomenko. Developing semantic vr-shops for e-commerce. *Virtual Reality*, 11(2-3):89–106, 2007.

[125] Olga De Troyer, Frederic Kleinermann, Bram Pellens, and Wesley Bille. Conceptual modeling for virtual reality. In John Grundy, Sven Hartmann, Alberto H. F. Laender, Leszek Maciaszek, and John F. Roddick, editors, *Tutorials, posters, panels and industrial contributions at the 26th International Conference on Conceptual Modeling - ER 2007*, volume 83 of *CRPIT*, pages 3–18, Auckland, New Zealand, 2007. ACS.

[126] Bianca Falcidieno and Michela Spagnuolo. A shape abstraction paradigm for modeling geometry and semantics. In *Computer Graphics International*, pages 646–. IEEE Computer Society, 1998.

[127] Bianca Falcidieno, Michela Spagnuolo, Pierre Alliez, E Quak, E Vavalis, and C Houstis. Towards the semantics of digital shapes: The aim@shape approach. In *EWIMT*, 2004.

[128] Steven Feiner, Blair Macintyre, and Dorée Seligmann. Knowledge-based augmented reality. *Commun. ACM*, 36(7):53–62, July 1993.

[129] Martin Fischbach, Dennis Wiebusch, Anke Giebler-Schubert, Marc Erich Latoschik, Stephan Rehfeld, and Henrik Tramberend. SiXton's curse - Simulator X demonstration. In Michitaka Hirose, Benjamin Lok, Aditi Majumder, and Dieter Schmalstieg, editors, *Virtual Reality Conference (VR), 2011 IEEE*, pages 255–256, 2011.

[130] Jakub Flotyński. Harvesting of semantic metadata from distributed 3d web content. In *Proceedings of the 6th International Conference on Human System Interaction (HSI), June 06-08, 2013, Sopot (Poland)*. IEEE, 2013.

[131] Jakub Flotyński. Semantic modelling of interactive 3d content with domain-specific ontologies. *Procedia Computer Science*, 35:531–540, 2014. 18th International Conference on Knowledge-Based and Intelligent Information & Engineering Systems.

[132] Jakub Flotyński, Jacek Dalkowski, and Krzysztof Walczak. Building multi-platform 3d virtual museum exhibitions with flex-vr. In *The 18th International Conference on Virtual Systems and Multimedia*, pages 391–398, Milan, Italy, September 2-5, 2012.

[133] Jakub Flotyński and Krzysztof Walczak. Attribute-based semantic descriptions of interactive 3d web content. In Leszek Kiełtyka, editor, *Information Technologies in Organizations - Management and Applications of Multimedia*, pages 111–138. Wydawnictwa Towarzystwa Naukowego Organizacji i Kierownictwa - Dom Organizatora, 2013.

[134] Jakub Flotyński and Krzysztof Walczak. Conceptual semantic representation of 3d content. *Lecture Notes in Business Information Processing: 16th International Conference on Business Information Systems, Poznań, Poland, 19 - 20 June, 2013*, 160:244–257, 2013.

[135] Jakub Flotyński and Krzysztof Walczak. Describing semantics of 3d web content with rdfa. In *The First International Conference on Building and Exploring Web Based Environments, Sevilla (Spain), January 27 - February 1, 2013*, pages 63–68. ThinkMind, 2013.

[136] Jakub Flotyński and Krzysztof Walczak. Microformat and microdata schemas for interactive 3d web content. In Maria Ganzha, Leszek Maciaszek, and Marcin Paprzycki, editors, *Proceedings of the 2013 Federated Conference on Computer Science and Information Systems Kraków, Poland, 8 - 11 September, 2013*, volume 1, pages 549–556. Polskie Towarzystwo Informatyczne, 2013.

[137] Jakub Flotyński and Krzysztof Walczak. Semantic modelling of interactive 3d content. In *Proceedings of the 5th Joint Virtual Reality Conference*, Paris, France, December 11-13, 2013.

[138] Jakub Flotyński and Krzysztof Walczak. Semantic multi-layered design of interactive 3d presentations. In *Proceedings of the Federated Conference on Computer Science and Information Systems*, pages 541–548, Kraków, Poland, September 8-11, 2013. IEEE.

[139] Jakub Flotyński and Krzysztof Walczak. Conceptual knowledge-based modeling of interactive 3d content. *The Visual Computer*, pages 1–20, August 2014.

[140] Jakub Flotyński and Krzysztof Walczak. Multi-platform semantic representation of 3d content. In *Proceedings of the 5th Doctoral Conference on Computing, Electrical and Industrial Systems*, Lisbon, Portugal, April 7-9 2014.

[141] Jakub Flotyński and Krzysztof Walczak. Semantic representation of multi-platform 3d content. *Computer Science and Information Systems*, 11, No 4, October 2014:1555–1580, 2014.

[142] Jakub Flotyński and Krzysztof Walczak. Ontology-based creation of 3d content in a service-oriented environment. Lecture Notes in Business Information Processing, International Conference on Business Information Systems. Springer Verlag, 2015, accepted for publication.

[143] Véronique Gaildrat. Declarative modelling of virtual environments, overview of issues and applications. In *International Conference on Computer Graphics and Artificial Intelligence (3IA), Athenes, Grece*, volume 10, pages 5–15, 2007.

[144] A. García-Rojas, F. Vexo, D. Thalmann, A. Raouzaiou, K. Karpouzis, and S. Kollias. 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, 2006.

[145] Michael Gosele, Wolfgang Stuerzlinger, et al. Semantic constraints for scene manipulation. In *in Proceedings Spring Conference in Computer Graphics' 99 (Budmerice, Slovak Republic*. Citeseer, 1999.

[146] GrammaTech. Halstead metrics. `http://www.grammatech.com/codesonar/workflow-features/halstead`, accessed March 24, 2015.

[147] Tom Gruber. Encyclopedia of database systems. `http://tomgruber.org/writing/ontology-definition-2007.htm`, accessed March 28, 2015.

[148] P Grussenmeyer, M Koehl, and M Nourel. 3d geometric and semantic modelling in historic sites, 1999.

[149] Mario Gutiérrez. Semantic virtual environments. 2005.

[150] Mario Gutiérrez, Alejandra García-Rojas, Daniel Thalmann, Frédéric Vexo, Laurent Moccozet, Nadia Magnenat-Thalmann, Michela Mortara, and Michela Spagnuolo. An ontology of virtual humans: Incorporating semantics into human shapes. *Vis. Comput.*, 23(3):207–218, February 2007.

[151] Mario Gutiérrez, Daniel Thalmann, and Frédéric Vexo. Semantic virtual environments with adaptive multimodal interfaces. In Yi-Ping Phoebe Chen, editor, *MMM*, pages 277–283. IEEE Computer Society, 2005.

[152] Mario Gutiérrez, Frédéric Vexo, and Daniel Thalmann. Semantics-based representation of virtual environments. *IJCAT*, 23(2-4):229–238, 2005.

[153] Pavel Halabala. Semantic metadata creation. In *Proceedings of 7th Central European Seminar on Computer Graphics CESCG*, pages 15–25, 2003.

[154] JungHyun Han, Ingu Kang, Chungmin Hyun, Jong-Sik Woo, and Young-Ik Eom. Multi-platform online game design and architecture. In *Proceedings of the 2005 IFIP TC13 international conference on Human-Computer Interaction*, pages 1116–1119, Springer Berlin Heidelberg, 2005.

[155] Guido Heumer, Malte Schilling, and Marc Erich Latoschik. Automatic data exchange and synchronization for knowledge-based intelligent virtual environments. In *Proceedings of the IEEE VR2005*, pages 43–50, 2005.

[156] Sylvia Irawati, Daniela Calderón, and Heedong Ko. Semantic 3d object manipulation using object ontology in multimodal interaction framework. In *Proceedings of the 2005 international conference on Augmented tele-existence*, pages 35–39. ACM, 2005.

[157] ISO. Iso/iec 15938-1:2002 - information technology – multimedia content description interface – part 1: Systems. `http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=34228`, accessed March 24, 2015.

[158] Jacek Jankowski and Stefan Decker. A dual-mode user interface for accessing 3d content on the world wide web. In *Proceedings of the 21st International World Wide Web Conference (WWW'12)*, pages 1047–1056. ACM, April 16-20 2012.

[159] Hao Jiang, Wenbin Xu, Tianlu Mao, Chunpeng Li, Shihong Xia, and Zhaoqi Wang. A semantic environment model for crowd simulation in multilayered complex environment. In *Proceedings of the 16th ACM Symposium on Virtual Reality Software and Technology*, pages 191–198. ACM, 2009.

[160] E. Kalogerakis, Stavros Christodoulakis, and Nektarios Moumoutzis. Coupling ontologies with graphics content for knowledge driven visualization. In *VR '06 Proceedings of the IEEE conference on Virtual Reality*, pages 43–50, Alexandria, Virginia, USA, March 25-29, 2006.

[161] Patrick Kapahnke, Pascal Liedtke, Stefan Nesbigall, Stefan Warwas, and Matthias Klusch. Isreal: An open platform for semantic-based 3d simulations in the 3d internet. In *International Semantic Web Conference (2)*, pages 161–176, 2010.

[162] Jassin Kessing, Tim Tutenel, and Rafael Bidarra. Designing semantic game worlds. In *Proceedings of the The Third Workshop on Procedural Content Generation in Games*, PCG'12, pages 2:1–2:9, New York, NY, USA, 2012. ACM.

[163] Frederic Kleinermann, Olga De Troyer, Christophe Creelle, and Bram Pellens. Adding semantic annotations, navigation paths and tour guides to existing virtual environments. In Theodor G. Wyeld, Sarah Kenderdine, and Michael J. Docherty, editors, *VSMM*, volume 4820 of *Lecture Notes in Computer Science*, pages 100–111. Springer, 2007.

[164] Frederic Kleinermann, Olga De Troyer, Haïthem Mansouri, Raul Romero, Bram Pellens, and Wesley Bille. Designing semantic virtual reality applications. In *In Proceedings of the 2nd INTUITION International Workshop, Senlis*, pages 5–10, 2005.

[165] Frederic Kleinermann, Haïthem Mansouri, Olga De Troyer, Bram Pellens, and Jesús Ibáñez-Martínez. Designing and using semantic virtual environment over the web. *IJVR*, 7(3):53–58, 2008.

[166] Hema S. Koppula, Abhishek Anand, Thorsten Joachims, and Ashutosh Saxena. Semantic labeling of 3d point clouds for indoor scenes. In J. Shawe-Taylor, R.S. Zemel, P.L. Bartlett, F. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 244–252. Curran Associates, Inc., 2011.

[167] Hamid Laga, Michela Mortara, and Michela Spagnuolo. Geometry and context for semantic correspondences and functionality recognition in man-made 3d shapes. *ACM Trans. Graph.*, 32(5):150:1–150:16, October 2013.

[168] Marc Erich Latoschik, Peter Biermann, and Ipke Wachsmuth. Knowledge in the loop: Semantics representation for multimodal simulative environments. In *Smart Graphics*, pages 25–39, 2005.

[169] Marc Erich Latoschik and Roland Blach. Semantic modelling for virtual worlds – a novel paradigm for realtime interactive systems? In *Proceedings of the ACM VRST 2008*, pages 17–20, 2008.

[170] Marc Erich Latoschik and Christian Fröhlich. Semantic reflection for intelligent virtual environments. In *IEEE Virtual Reality Conference 2007*, pages 305–306, Charlotte, USA, March 10-14, 2007.

[171] Marc Erich Latoschik and Christian Fröhlich. Towards intelligent vr: Multi-layered semantic reflection for intelligent virtual environments. Proceedings of the Graphics and Applications GRAPP 2007, pages 249–259, 2007.

[172] Marc Erich Latoschik and Malte Schilling. Incorporating vr databases into ai knowledge representations: A framework for intelligent graphics applications. In *Proceedings of the Sixth IASTED International Conference on Computer Graphics and Imaging*, pages 79–84. Press, 2003.

[173] Marc Erich Latoschik and Henrik Tramberend. Simulator X: A Scalable and Concurrent Software Platform for Intelligent Realtime Interactive Systems. In *Proceedings of the IEEE VR 2011*, 2011.

[174] O Le Roux, V Gaildrat, and R Caube. Constraint satisfaction techniques for the generation phase in declarative modeling. In *Geometric modeling: techniques, applications, systems and tools*, pages 193–215. Springer, 2004.

[175] Daniel Lipkin. Integrating xml and vrml: A technical discussion. `http://xml.coverpages.org/lipkin-vrmlxml.html`, accessed March 24, 2015.

[176] John Wylie Lloyd. Foundations of logic programming. 1987.

[177] M Lorenzini. Semantic approach to 3d historical reconstruction. In *Proceedings of the 3rd ISPRS International Workshop 3D-ARCH 2009:" 3D Virtual Reconstruction and Visualization of Complex Architectures" Trento, Italy, 25-28 February 2009*, 2009.

[178] Michael Luck and Ruth Aylett. Applying artificial intelligence to virtual reality: Intelligent virtual environments. *Applied Artificial Intelligence*, 14(1):3–32, 2000.

[179] Jean-Luc Lugrin. *Alternative Reality and Causality in Virtual Environments*. PhD thesis, University of Teesside, Middlesbrough, United Kingdom, 2009.

[180] Jean-Luc Lugrin and Marc Cavazza. Making sense of virtual environments: Action representation, grounding and common sense. In *Proceedings of the 12th international conference on Intelligent user interfaces*, IUI '07, pages 225–234, New York, NY, USA, 2007. ACM.

[181] Haithem Mansouri. *Using Semantic Descriptions for Building and Querying Virtual Environments*. PhD thesis, Vrije Universiteit Brussel, 2004-2005.

[182] Caroline M. Mendes, Dyego R. Drees, Luciano Silva, and Olga R. Bellon. Interactive 3d visualization of natural and cultural assets. In *Proceedings of the second workshop on eHeritage and digital art preservation*, pages 49–54, Firenze, Italy, October 25-29, 2010.

[183] M. Mortara, G. Patané, and M. Spagnuolo. From geometric to semantic human body models. *Comput. Graph.*, 30(2):185–196, April 2006.

[184] Franco Niccolucci and Andrea D'Andrea. An ontology for 3d cultural objects. In Marinos Ioannides, David B. Arnold, Franco Niccolucci, and Katerina Mania, editors, *VAST*, pages 203–210. Eurographics Association, 2006.

[185] OMG. Unified modeling language (uml), accessed March 24, 2015.

[186] Oracle. Java3d. `http://www.oracle.com/`, accessed March 24, 2015.

[187] O'Reilly. Metrics for xml projects. `http://www.oreillynet.com/xml/blog/2006/05/`.

[188] Karsten Otto. Semantic virtual environments. In *Special interest tracks and posters of the 14th international conference on World Wide Web*, pages 1036–1037, Chiba, Japan, May 10-14, 2005.

[189] Karsten Otto. The semantics of multi-user virtual environments. In *Proceedings of the Workshop towards Semantic Virtual Environments*, 2005.

[190] L. Papaleo, R. Albertoni, S Marini, and F Robbiano. An ontology-based approach to acquisition and reconstruction. In *Workshop towards Semantic Virtual Environment, Villars, Switzerland*, 2005.

[191] L. Papaleo, L. De Floriani, J. Hendler, and A. Hui. Towards a semantic web system for understanding real world representations. In *Proceedings of the Tenth International Conference on Computer Graphics and Artificial Intelligence*, 2007.

[192] Changhoon Park, TaeSeok Jin, Michitaka Hiroseo, and Heedong Ko. A framework for vr application based on spatial, temporal and semantic relationship. In *Virtual Reality*, pages 329–337. Springer, 2007.

[193] Clark & Parsia. Pellet: Owl 2 reasoner for java. `http://clarkparsia.com/pellet/`, accessed March 24, 2015.

[194] Cameron D Pelkey and Jan M Allbeck. Populating semantic virtual environments. *Computer Animation and Virtual Worlds*, 25(3-4):405–412, 2014.

[195] Bram Pellens. *A Conceptual Modelling Approach for Behaviour in Virtual Environments using a Graphical Notation and Generative Design Patterns*. PhD thesis, Vrije Universiteit Brussel, 2006-2007.

[196] Bram Pellens, Olga De Troyer, Wesley Bille, Frederic Kleinermann, and Raul Romero. An ontology-driven approach for modeling behavior in virtual environments. In R. Meersman, Z. Tari, and P. Herrero, editors, *Proceedings of On the Move to Meaningful Internet Systems 2005: Ontology Mining and Engineering and its Use for Virtual Reality (WOMEUVR 2005) Workshop*, number 3762 in Lecture Notes in Computer Science, pages 1215–1224, Agia Napa, Cyprus, 2005. Springer-Verlag, Springer-Verlag.

[197] Bram Pellens, Olga De Troyer, and Frederic Kleinermann. Codepa: a conceptual design pattern approach to model behavior for x3d worlds. In *Proceedings of the 13th International Symposium on 3D web technology*, pages 91–99, Los Angeles, August 09-10, 2008.

[198] Bram Pellens, Frederic Kleinermann, and Olga De Troyer. Intuitively specifying object dynamics in virtual environments using vr-wise. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, VRST '06, pages 334–337, New York, NY, USA, 2006. ACM.

[199] Bram Pellens, Frederic Kleinermann, and Olga De Troyer. A development environment using behavior patterns to facilitate building 3d/vr applications. In *Proceedings of the Sixth Australasian Conference on Interactive Entertainment*, IE '09, pages 8:1–8:8. ACM, 2009.

[200] Fabio Pittarello and Alessandro De Faveri. Semantic description of 3d environments: A proposal based on web standards. In *Proceedings of the Eleventh International Conference on 3D Web Technology*, Web3D '06, pages 85–95, New York, NY, USA, 2006. ACM.

[201] Dimitri Plemenos. Using artificial intelligence techniques in computer graphics. In *International Conference Graphicon*, 1999.

[202] Dimitri Plemenos and Georgios Miaoulis. *Artificial intelligence techniques for computer graphics*, volume 159. Springer, 2008.

[203] Anshuman Razdan, Jeremy Rowe, Matthew Tocheri, and Wilson Sweitzer. Adding semantics to 3d digital libraries. In EePeng Lim, Schubert Foo, Christopher S. G. Khoo, Hsinchun Chen, Edward A. Fox, Shalini R. Urs, and Costantino Thanos, editors, *ICADL*, volume 2555 of *Lecture Notes in Computer Science*, pages 419–420. Springer, 2002.

[204] R Reffat. Semantic-based virtual design environments for architecture. *Proceedings of Education of Computer Aided Architectural Design in Europe (eCAADe)*, 2003.

[205] Gerhard Reitmayr and Dieter Schmalstieg. Semantic world models for ubiquitous augmented reality. In *Proceedings of Workshop towards Semantic Virtual Environments' (SVE) 2005*, 2005.

[206] Aristides G. Requicha. Representations for rigid solids: Theory, methods, and systems. *ACM Comput. Surv.*, 12(4):437–464, December 1980.

[207] Francesco Robbiano, Marco Attene, Michela Spagnuolo, and Bianca Falcidieno. Part-based annotation of virtual 3d shapes. *2013 International Conference on Cyberworlds*, 0:427–436, 2007.

[208] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003.

[209] Tianjia Shao, Weiwei Xu, Kun Zhou, Jingdong Wang, Dongping Li, and Baining Guo. An interactive approach to semantic modeling of indoor scenes with an rgbd camera. *ACM Transactions on Graphics*, 31(6):136:1–136:11, November 2012.

[210] Michela Spagnuolo and Bianca Falcidieno. *The Role of Ontologies for 3D Media Applications*. Springer, 2008.

[211] Michela Spagnuolo and Bianca Falcidieno. 3d media and the semantic web. *IEEE Intelligent Systems*, 24(2):90–96, 2009.

[212] Klaas Tack, Gauthier Lafruit, Francky Catthoor, and Rudy Lauwereins. Platform independent optimisation of multi-resolution 3d content to enable universal media access. *The Visual Computer*, 22, Issue 8:577–590, August 2006.

[213] Thanh-Hai Trinh, Ronan Querrec, Pierre De Loor, and Pierre Chevaillier. Ensuring semantic spatial constraints in virtual environments using uml/ocl. In *Proceedings of the 17th ACM Symposium on Virtual Reality Software and Technology*, pages 219–226. ACM, 2010.

[214] Tim Tutenel, Rafael Bidarra, Ruben M Smelik, and Klaas Jan De Kraker. The role of semantics in games and simulations. *Computers in Entertainment (CIE)*, 6(4):57, 2008.

[215] Tim Tutenel, Ruben Michaël Smelik, Rafael Bidarra, and Klaas Jan de Kraker. Using semantics to improve the design of game worlds. In *AIIDE*, 2009.

[216] Tim Tutenel, Ruben Michaël Smelik, Rafael Bidarra, and Klaas Jan de Kraker. A semantic scene description language for procedural layout solving problems. *AIIDE*, 10:1–6, 2010.

[217] Tim Tutenel, Ruben Michaël Smelik, Ricardo Lopes, Klaas Jan de Kraker, and Rafael Bidarra. Generating consistent buildings: a semantic approach for integrating procedural techniques. *Computational Intelligence and AI in Games, IEEE Transactions on*, 3(3):274–288, 2011.

[218] Luc Van Gool, Bastian Leibe, Pascal Müller, Maarten Vergauwen, and Thibaut Weise. 3d challenges and a non-in-depth overview of recent progress. In *3DIM*, pages 118–132, 2007.

[219] Lode Vanacken, Chris Raymaekers, and Karin Coninx. Introducing semantic information during conceptual modelling of interaction for virtual environments. In *Proceedings of the 2007 Workshop on Multimodal Interfaces in Semantic Interaction*, WMISI '07, pages 17–24, New York, NY, USA, 2007. ACM.

[220] George Vasilakis, Alejandra García-Rojas, Laura Papaleo, Chiara Eva Catalano, Francesco Robbiano, Michela Spagnuolo, Manolis Vavalis, and Marios Pitikakis. Knowledge-based representation of 3d media. *International Journal of Software Engineering and Knowledge Engineering*, 20(5):739–760, 2010.

[221] W3C. Building the web of data. http://www.w3.org/2013/data/, accessed March 24, 2015.

[222] W3C. Iso/iec 14772-1:1997. the virtual reality modeling language. http://www.web3d.org/x3d/specifications/, accessed March 24, 2015.

[223] W3C. Iso/iec 19775-1:2008. extensible 3d (x3d) (2008). http://web3d.org/x3d/specifications/, accessed March 24, 2015.

[224] W3C. Owl. http://www.w3.org/2001/sw/wiki/OWL, accessed March 24, 2015.

[225] W3C. Rdf. http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/, accessed March 24, 2015.

[226] W3C. Rdfs. http://www.w3.org/TR/2000/CR-rdf-schema-20000327/, accessed March 24, 2015.

[227] W3C. Swrl. http://www.w3.org/Submission/SWRL/, accessed March 24, 2015.

[228] Krzysztof Walczak. *Database Modeling of Virtual Reality*. PhD thesis, Technical University of Gdańsk, 2001.

[229] Krzysztof Walczak. Beh-vr: modeling behavior of dynamic virtual reality contents. In *Interactive Technologies and Sociotechnical Systems*, pages 40–51. Springer, 2006.

[230] Krzysztof Walczak. Flex-vr: configurable 3d web applications. In *Proceedings of the Conference on Human System Interactions*, pages 135–140. IEEE, 2008.

[231] Krzysztof Walczak. *Configurable Virtual Reality Applications*. Poznań, 2009.

[232] Krzysztof Walczak. Building configurable 3d web applications with flex-vr. In Wojciech Cellary and Krzysztof Walczak, editors, *Interactive 3D Multimedia Content*, pages 103–136. Springer, 2012.

[233] Krzysztof Walczak. Dynamic database modeling of 3d multimedia content. In Wojciech Cellary and Krzysztof Walczak, editors, *Interactive 3D Multimedia Content*, pages 55–102. Springer, 2012.

[234] Krzysztof Walczak and Wojciech Cellary. X-vrml for advanced virtual reality applications. *Computer*, 36(3):89–92, March 2003.

[235] Krzysztof Walczak, Wojciech Cellary, Jacek Chmielewski, Mirosław Stawniak, Sergiusz Strykowski, Wojciech Wiza, Rafał Wojciechowski, and Adam Wojtowicz. An architecture for parameterised production of interactive tv contents. In *International workshopon systems, signals and image processing, ambient multimedia*, pages 465–468, 2004.

[236] Krzysztof Walczak, Wojciech Cellary, and Martin White. Virtual museum exhibitions. *Computer*, 39(3):93–95, March 2006.

[237] Krzysztof Walczak, Jacek Chmielewski, Miroslaw Stawniak, and Sergiusz Strykowski. Extensible metadata framework for describing virtual reality and multimedia contents. In Hamza, MH, editor, *Proc. of the IASTED Int. Conf. on Databases and Applications*, pages 168–175. Int Assoc Sci & Technol Dev; IASTED Tech Comm Databases, 2006. IASTED Int. Conf. on Databases and Applications, Innsbruck, Austria, Feb 14-16, 2006.

[238] Krzysztof Walczak and Jakub Flotyński. On-demand generation of 3d content based on semantic meta-scenes. In *Lecture Notes in Computer Science; Augmented and Virtual Reality; First International Conference, AVR 2014, Lecce, Italy, September 17-20, 2014*, pages 313–332. Springer International Publishing, 2014.

[239] Krzysztof Walczak, Dariusz Rumiński, and Jakub Flotyński. Building contextual augmented reality environments with semantics. In *Proocedings of the 20th International Conference on Virtual Systems & Multimedia, Hong Kong, 9-12 September*, 2014.

[240] Krzysztof Walczak, Rafał Wojciechowski, and A. Wójtowicz. Interactive production of dynamic 3d sceneries for virtual television studio. In *The 7th Virtual Reality IC VRIC - Laval Virtual 2005*, pages 167–177, April 2005.

[241] C. M. Whissel. Emotion: Theory, research and experience. In *The dictionary of affect in language*, volume 4, New York, 1989.

[242] Martin White, Nicholaos Mourkoussis, Joe Darcy, Panos Petridis, Fotis Liarokapis, Paul F. Lister, Krzysztof Walczak, Rafal Wojciechowski, Wojciech Cellary, Jacek Chmielewski, Miroslaw Stawniak, Wojciech Wiza, Manjula Patel, James Stevenson, John Manley, Fabrizio Giorgini, Patrick Sayd, and François Gaspard.

Arco - an architecture for digitization, management and presentation of virtual exhibitions. In *Computer Graphics International*, pages 622–625. IEEE Computer Society, 2004.

[243] Dennis Wiebusch and Marc Erich Latoschik. Enhanced Decoupling of Components in Intelligent Realtime Interactive Systems using Ontologies. In *Software Engineering and Architectures for Realtime Interactive Systems (SEARIS), proceedings of the IEEE Virtual Reality 2012 workshop*, 2012.

[244] Rafal Wojciechowski, Krzysztof Walczak, Martin White, and Wojciech Cellary. Building virtual and augmented reality museum exhibitions. In *Proceedings of the Ninth International Conference on 3D Web Technology*, Web3D '04, pages 135–144, New York, NY, USA, 2004. ACM.

[245] Xuehan Xiong and Daniel Huber. Using context to create semantic 3d models of indoor environments. In *BMVC*, pages 1–11, 2010.

[246] Ken Xu, James Stewart, and Eugene Fiume. Constraint-based automatic placement for scene composition. In *Graphics Interface*, volume 2, 2002.

[247] Chih-Hao Yu. Semantic annotation of 3d digital representation of cultural artefacts, accessed March 24, 2015.

[248] Theodore Zahariadis, Petros Daras, and Isidro Laso-Ballesteros. Towards future 3d media internet. *NEM Summit*, pages 13–15, 2008.

[249] Lamia Abo Zaid, Frederic Kleinermann, and Olga De Troyer. Applying semantic web technology to feature modeling. In *Proceedings of the 2009 ACM Symposium on Applied Computing*, SAC '09, pages 1252–1256. ACM, 2009.

[250] Youyi Zheng, Daniel Cohen-Or, and Niloy J Mitra. Smart variations: Functional substructures for part compatibility. *Computer Graphics Forum*, 32(2pt2):195–204, 2013.

List of Figures

List of Tables

Listings