GDAŃSK UNIVERSITY OF TECHNOLOGY
FACULTY OF ELECTRONICS, TELECOMMUNICATIONS AND INFORMATICS

The author of the PhD dissertation: Dariusz Rumiński
Scientific discipline: Computer science

DOCTORAL DISSERTATION

Title of PhD dissertation:
Semantic modeling of contextual augmented reality environments

Title of PhD dissertation (in Polish):
Semantyczne modelowanie kontekstowych środowisk wzbogaconej rzeczywistości

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, 2018

Contents

1. Introduction

Augmented reality (AR) technology enables superimposing computer-generated content, such as interactive 2D and 3D multimedia objects, in real time, on a view of real-world objects [23]. Widespread use of AR technology has been enabled in the recent years by remarkable progress in consumer-level hardware performance, in particular, in the computational and graphical performance of computing hardware and quickly growing bandwidth of cellular networks. The progress is particularly visible in the domain of mobile devices, such as smartphones, tablets, and wearables. These devices are nowadays equipped with multi-core processors, large amounts of memory, high-quality displays and multi-modal interaction devices, such as accelerometers, gyroscopes, cameras, microphones, and GPS sensors. Mobile devices became general-purpose computing platforms well suited for the deployment of various kinds of multimedia applications. Moreover, rapid growth in the available bandwidth of wireless networks, which is now sufficient to deliver large amounts of data required by interactive 3D multimedia applications, makes the use of mobile devices for this kind of applications even more appealing. Last but not least, the progress is evident in the domain of AR and VR startups that develop augmented and virtual reality technologies. According to [40], AR/VR startups raised a record over $3.6 billion funding from venture capital corporates within 12 months to the end of Q1 2018.

Augmented reality, with its potential to blend real and virtual objects, creates new opportunities for building interactive and engaging applications. Education [29, 158, 161], entertainment [75, 79, 99], medicine [78, 97, 134], and cultural heritage [62, 88, 157] are examples of application domains in which AR-based systems are increasingly being used.

Currently, there is a large variety of AR applications developed by both researchers and commercial companies, and new applications appear frequently. As a rule, such applications are built independently for specific purposes. This situation forces end-users to continuously install, update, and uninstall multiple – in most cases relatively short-life AR applications – instead of having one long-life universal application that would act as a kind of "AR browser". Nowadays, AR content, interfaces, functionality are fragmented between independent AR programs, in such a way, that end users have to install each of them individually – taking into account their compatibility with the operating system and an end-user's device. Also, the current AR applications do not enable to experience AR presentations in a continuous and contextual manner, i.e., regardless where the user is located (indoor or outdoor), current time and date, what kind of platform and device type are used, and – what is very important – taking into account user preferences and needs. To facilitate the use of AR in various application domains and for information visualization in general – independently of a specific application, platform or device being used, and also to be widely used and accepted by end users, new models and methods of building AR content and applications are required.

The currently available AR tools range from general purpose computer vision and graphics libraries and frameworks [5, 17, 44, 52, 53, 58, 63, 70, 82, 92, 110, 125, 130], requiring advanced programming skills to develop applications, to easy-to-use point-and-click packages for computer and mobile devices

[4, 8, 11, 61, 82, 118, 122, 128, 168], enabling creation of simple AR presentations. These tools provide functionality for manual authoring of AR presentations – either through programming or visual design.

The real challenge lies in building ubiquitous contextual AR environments, in which AR presentations are not dependent on a specific application, platform or device, but elements forming AR experiences are independently provided in real time by the available external sources. In such system, a key mechanism responsible for automatic selection and composition of AR resources should be based on the user's context. The current context can reflect what kind of content a user wants to experience, taking also into account time, date, indoor or outdoor location, the user's device type and its capabilities. To enable widespread adoption of AR in various application domains, not only end users must have simple ways of accessing AR content on-site, but also the presentation designers must have intuitive and easy-to-use methods of creating AR presentations. Such a universal AR system that dynamically generates personalized AR presentations, based on context and various multimedia content and data providers, requires solving the problem of describing, searching, interpreting, combining and presenting independent resources that jointly form interactive AR presentations. Semantic web techniques have the potential to provide a suitable solution, however, as of yet, their application in the context of AR has not yet been sufficiently explored.

The research on semantic internet has been initiated by T. Berners-Lee and the W3C consortium [24] and led to the development of standards such as RDF [147], RDFS [147], and OWL [151]. The standards were designed to enable evolution of the web towards a distributed semantic database linking structured documents. This type of semantic description, by precisely defining the meaning of individual data elements and relationships between them – in a manner that is understandable for both people and computers – allows achieving an entirely new quality in building augmented reality applications. Semantic web techniques allow efficient description, search, retrieval and presentation of multimedia content and data. Moreover, it is also possible to apply reasoning methods in order to derive new knowledge that is not explicitly stated. At the core of semantic web is decoupling applications from data through the use of an abstract model for knowledge representation. This abstract model allows releasing bilateral constraints on applications and data, letting both to evolve independently. As a consequence of using such model, any application that understands the model can consume any data source using this model [24].

The idea of connecting semantic web with AR dates in literature back to 2005 [113] and research on this subject is still underway. In most cases, researches have focused on developing location-based AR interfaces capable to overlay 2D annotations onto a view of real-world object additionally enriched with data coming from specific semantic linked data servers [12, 26, 64, 91, 100, 124, 145, 167]. Such systems do not permit a user to fully experience augmented reality presentations based on rich, multimedia 3D content which could be reused in many different use cases. Moreover, the above-mentioned systems use specific semantic web ontologies to specific applications. Thus, to create another AR application enhanced with semantic web techniques, there would be a need to develop yet another specific semantic data model. Last, but not least significant drawback of previous works in this field is lack of any quantitative evaluation of the developed solutions that would prove their applicability and efficiency in case of large-scale contextual distributed environments.

In this dissertation a new approach to the problem of modeling augmented reality applications is presented. This approach, called *Contextual Augmented Reality Environment* (CARE), goes beyond the current state of the art in modeling augmented reality environments by providing uniform cross-application contextual access to distributed semantically-described AR content and services. The novelty of CARE lies in avoiding fragmentation of AR functionality between multiple independent

applications and simplifying integration of various information sources into a unified, contextual, and personalized AR interface. CARE supports operations such as describing, designing, searching, interpreting, combining and presenting content and data that jointly form personalized AR presentations. In CARE, augmented reality presentations are built on the basis of the user's context, which includes such elements as user preferences, time, date, outdoor or indoor location, the user's device type, and its capabilities.

The CARE approach consists of four key elements:

1. An architecture of distributed AR services that supports semantic modeling and building of contextual AR presentations for a large number of users. The presented architecture consists of two client applications and multiple server modules that combined enable designers and end users to model and explore a contextual AR environment with the use of semantics.
2. The *Semantic Augmented Reality Ontology* (SARO), which encompasses a set of concepts and properties describing various aspects of the user's context and distributed resources forming AR presentations. SARO is based on semantic web standards (RDF, RDFS, and OWL 2). Moreover, certain elements of context are built on top of well-known standards, such as Geography Markup Language Encoding Standard (GML) and W3C OWL-time ontology. The model is used to build ubiquitous dynamic augmented reality environments based on semantically described resources that are contextually provided by AR Service providers.
3. The *Semantic Discovery and Matching Method* (SDMM). The method is responsible for composing contextual augmented reality presentations based on multiple distributed AR resources taking into account user's context. SDMM searches AR resources and service providers through semantic knowledge base structured with the SARO ontology. SDMM allows knowledge discovery in the process of building contextual AR presentations. SDMM can be used in two modes. In the first mode the SDMM returns only the first available AR resource within the user's context. In this case, the goal is to return as quickly as possible usable results, which may not be optimal. In the second mode the SDMM is set to return optimal results. This configuration returns all results related to the geographical location of a user, but it is more time-consuming.
4. A new high-level declarative language, called *Contextual Augmented Reality Language* (CARL), which is used as an application protocol. It enables modeling AR presentations whose elements come from diverse and distributed content- and data service providers. The CARL language is designed to support dynamic composition of complex interactive AR presentations.

The thesis of this dissertation is formulated as follows:

The CARE approach enables efficient modeling of large-scale contextual distributed augmented reality environments.

The dissertation is composed of seven chapters and is organized as follows.

Chapter 2 provides an overview of the state of the art of AR including first AR display presented in 1968, two well-known AR definitions cited in the literature, and a review of noteworthy AR programs applied in multiple application domains is presented.

Chapter 3 discusses AR development tools that can be used to create various kinds of augmented reality applications – from low-level software libraries and frameworks requiring programming skills to use them, to simple and easy-to-use visual authoring tools for non-technical users. In addition, AR declarative languages used for designing AR applications, are discussed. Also, this chapter explores the role of semantic web and how it was applied to AR. The chapter introduces semantic web standards and

popular serialization formats with examples. After that, applications of the semantic web technology in augmented reality systems is presented. In the end, the limitations of existing methods of modeling AR environments are discussed.

Chapter 4 presents the concept of the *Contextual Augmented Reality Environment* approach including a formal model that is based on set theory and functions. Also, the main elements of CARE are presented: the architecture of distributed AR services, including two client applications and multiple server modules; the *Semantic Augmented Reality Ontology* that enables to model independent AR service providers, resources, and contexts, and links all these elements to form a ubiquitous dynamic AR environment; the *Semantic Discovery and Matching Method* that is responsible for selecting semantically described data meeting criteria of a user's context; and a novel high-level language, called *Contextual Augmented Reality Language*, which constitutes the application protocol between the client and the server sides by allowing description of dynamic AR presentations.

Chapter 5 provides an overview of the implementation of the CARE system. First, the client-server architecture of CARE is presented, including data flow diagrams between the main system elements. Then, the client-side as well as the server-side components are characterized. Next, description of two client-side applications is provided – *CARE Modeler* implemented as an extension to the popular Unity3D IDE and used for modeling of CARE environments; *BrowsAR* – an Android-based AR browser used for experiencing contextual AR presentations. Further, server-side components are presented: Java EE-based application server *Semantic Augmented Reality Middleware* – responsible for providing semantic search and multiple RESTful web services offering AR resources that can be dynamically composed to build diverse-contextualized AR presentations. Finally, real-world use-cases of AR environments modeled with CARE are presented.

Chapter 6 describes qualitative and quantitative evaluation of the CARE approach and the obtained results. A qualitative user study has been performed to evaluate usefulness and easiness of use of the *CARE Modeler* and the *BrowsAR* applications, while modeling and exploring contextual augmented reality environment. The design of the study, characteristics of participants that took part in the study, and the collected results are presented followed by a discussion. Further, quantitative evaluation of the *Semantic Augmented Reality Middleware* performance is described. Load testing was performed under various conditions to verify how the *Semantic Augmented Reality Middleware* – in particular the *Search Service* – behaves in case when using different size knowledge bases. Moreover, the completion time of the SDMM method was measured to compare the method's performance versus *Search Service* response time. The description of quantitative evaluation covers design of the study, characteristics of the designed test plan, procedures and environmental setup, followed by the presentation of the performance results. Finally, discussion of the obtained results is provided.

Chapter 7 concludes the dissertation. The main contribution and achievements are discussed, and the possible directions of future research and development activities are indicated.

2. Augmented Reality Environments

In this chapter, a summary of the state of the art in the domain of augmented reality is provided. To begin with, the concept of augmented reality and the first known AR display, are presented. Next, definitions of augmented reality, which are often cited in literature, are discussed. The chapter then presents prominent application domains in which AR technology is used, including education, remote collaboration, gaming, medicine, cultural heritage, automotive, designing domains, and visualization of geo-localized data.

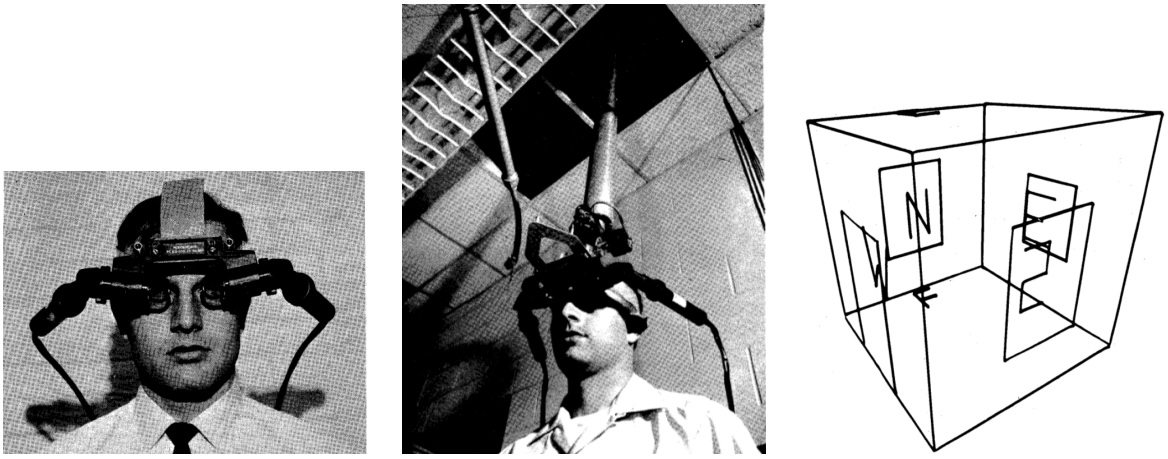
2.1. The concept of augmented reality

Augmented reality (AR) is a field of computer science that concerns computer vision-based technologies enabling superimposing rich computer-generated content – such as 2D and 3D multimedia objects – in real time, on a view of real objects.

The first AR display described in the literature dates back to 1968 [137]. That year, Ivan Sutherland introduced a head-mounted three-dimensional display (HMD) that was capable of blending transparent wire frame line drawings onto a user's view, in real time (Fig. 2.1). Sutherland's solution projected a simple wire frame cube with additional lines drawn on each wall representing compass directions (Fig. 2.1c) with the use of the HMD device. This work was a first step towards implementation of "The ultimate display" idea [136], in which the author described his vision of the future human-computer interface technology: *"The ultimate display would, of course, be a room within which the computer can control the existence of matter. A chair displayed in such a room would be good enough to sit in. Handcuffs displayed in such a room would be confining, and a bullet displayed in such a room would be fatal. With appropriate programming such a display could literally be the Wonderland into which Alice walked."* Sutherland's works gave innumerable inspirations to the next generations, on how user's surrounding can be augmented and computer vision technology integrated into the physical world.

Although Sutherland invented the first AR display, the term "Augmented Reality" became popular in the scientific community only after the publication of a special issue of the ACM Communications journal in 1993 devoted to the subject [30]. Below, definitions of augmented reality, that are often cited in literature, are given.

Milgram et al. introduced the concept of *Reality-Virtuality (RV) continuum*, which distinguishes augmented reality (AR) and virtual reality (VR) environments [94]. As shown in Figure 2.2, on the left side of the continuum is a physical environment consisting solely of real objects with no synthetic objects at all. On the right side of the continuum is a fully synthetic virtual environment, consisting purely of virtual objects with no information from the physical world being presented. Mixed Reality was defined as a class of systems that is located somewhere in between, since it is a combination of the two. In the continuum, to the right of the real environment – AR is situated, in which the real environment is enhanced with computer-generated content. Moving further to the right in the continuum, the augmented



(a) The head-mounted display headset. (b) The head position sensor in use. (c) The projection of a room as seen outside.

Figure 2.1: The first published augmented reality system enabling overlaying wire frame on a view of the user [137].

virtuality (AV) is located, where presentation of elements of the physical environment is added to a fully immersive virtual environment.

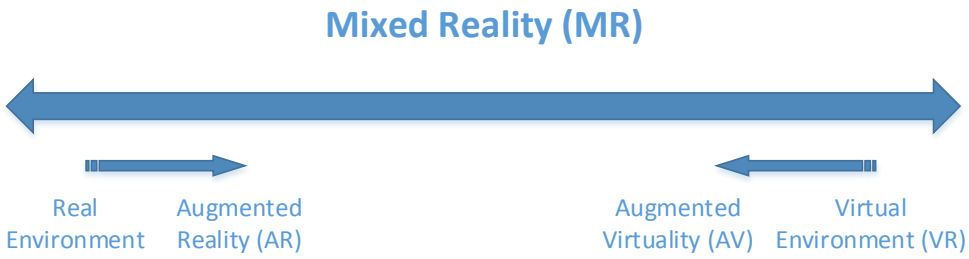


Figure 2.2: The Reality-Virtuality continuum concept [94].

In 1997, Ronald Azuma defined augmented reality as any system that meets the following three criteria [23]:

1) it combines real and virtual;
2) is interactive in real time;
3) is registered in 3D.

The first and the third criteria require integration of virtual objects into a real 3D environment. The definition does not specify which specific presentation technology should be used to meet the requirement – besides head mounted displays (HMDs) it is also applicable to use monitors, projectors, smartphones, wearable glasses, etc. The second requirement, which extends Milgram's et al. definition, means that a user can interact with virtual objects in real time. For instance, non-interactive media such as cinema movies (e.g., "Jurassic Park") are not interactive, e.g., a user cannot experience a synthetic model of a Tyrannosaurus Rex from different perspectives in real time. To meet the third requirement, the system needs to have information of the scene and observer's positions to appropriately overlay virtual objects. This property has to be dynamically updated when a scene or an observer move.

2.2. Application domains of AR

Augmented reality, with its potential to blend real and virtual objects, creates new opportunities for visualization of various kinds of contextual information. E-commerce, education, remote collaboration, entertainment, medicine, cultural heritage, automotive, designing, visualization of content and data, are prominent examples of application domains in which AR-based systems are increasingly being used. Below selected application domains of AR are discussed.

2.2.1. E-commerce

The *Magic Mirror* concept is an augmented reality system, consisting of a camera and a display device, that act as a mirror, in which a user sees a reflection of oneself enriched with virtual objects [28]. The paradigm has been already used to overlay virtual shirts [31, 42, 105], shoes [43], glasses [32], and knight's armors [46] onto the user's view. An example of the AR application implementing the *Magic Mirror* concept is presented in Figure 2.3, in which the user tries on the virtual glasses before buying the real ones. Such solutions enable end users to do shopping in a convenient way from home.

2.2.2. Education

Researchers have recognized a great value of AR in learning systems. For instance, one of the important advantages of using AR in education is freedom of experimentation, i.e., students can repeat experiments as many times as they want with no cost and no risk to loose health or laboratory materials [161]. In [39], Dede argued that AR is one of "next generation" pedagogical media promoting learning quality. The following research studies have been conducted regarding the usage of AR-based learning systems.

Wojciechowski and Cellary evaluated learners' attitude toward using the AR-based ARIES system in the context of chemistry learning [161]. With ARIES, students are able to conduct chemical experiments in person, using virtual counterparts of real laboratory equipment and chemicals. Figure 2.4 depicts students conducting AR chemical experiment.

Authors empirically examined the effects on using an AR interface while conducting chemical experiments. The results showed that perceived enjoyment and perceived usefulness had an effect on students' attitude toward using image-based AR environment. However, collected data demonstrated that perceived enjoyment was a much more significant factor than perceived usefulness, which essentially

Figure 2.3: The Magic Mirror concept in use.

Figure 2.4: AR chemistry experiment using ARIES [161].

influenced on the students' motivation to use the ARIES in the learning process. Furthermore, the authors suggest that the use of AR environments during lessons could provide additional motivation for students in order to improve their learning of chemistry. Moreover, authors argued that replacement of the real laboratory resources with their virtual counterparts makes it possible to achieve significant financial savings for educational institutions. The chemical AR installation takes up much less space than typical physical workbench for conducting chemical experiments. Moreover, it does not require any special costly chemistry laboratory infrastructure.

Mirracle is an AR system for teaching human anatomy which takes advantage of the *Magic Mirror* concept [28]. The system uses a depth camera to track the pose of a user standing in front of a large display – as presented in Figure 2.6. The system overlays computerized tomography (CT) scans onto a user's view, creating an illusion that the user looks into his/her body. Furthermore, *Mirracle* can overlay 3D models of organs, textual information, and pictures of human anatomy. A user interacts with anatomical objects using hand gestures to control human body visualization.

Authors presented the *Mirracle* system during the open day of a hospital and in a school. Primary findings show that AR in-situ visualization of human body organ was very attractive to use – especially for children.

Figure 2.5: Visualization of CT data on the user (Mirracle) [28].

2.2.3. Remote collaboration

Nowadays, with the increasingly widespread availability of collaboration tools, including telephones, email, video conferencing system, shared repositories, and community platforms, remote collaboration became a part of everyday life for many people. Researchers investigated also the possibilities of bringing AR technology to remote collaboration systems. One of the notable AR systems, that gave impact and inspiration to the researchers on the next decade, was presented by Kato and Billinghurst in [71]. Authors developed an augmented reality video conferencing system in which images of remote collaborators are overlaid on virtual monitors and can be freely placed in user's surrounding. With the use of this system, end users are able to collaboratively view and interact with synthetic content using a shared virtual whiteboard. Additionally, the authors also evaluated accuracy of marker detection and HMD calibration methods. It is worth to mention that these methods are extensively used in ARToolKit – a well-known framework used for building AR applications [70].

Regenbrecht et al. presented a video conferencing system (*cAR/PE!*), where users are able to communicate over a network in an environment simulating a traditional face-to-face meeting [111]. Authors presented a prototype which visualizes live video streams of the participants arranged around a virtual table with spatial sound support. Spatial sound driven by headphones audio hardware was used to indicate different user positions. The participants' task was to decide on the most aesthetic out of five car models placed on one side of the virtual meeting room. After completing the task, participants were asked to fill-in a questionnaire. The *cAR/PE!* system was rated as easy to use and overall user satisfaction was good. Moreover, a method of exchanging information verbally was also rated as satisfactory.

Another solution that enables users to collaborate remotely using hands-free video calling, image sharing, and drawing virtual annotations was presented by Microsoft [93]. With the use of remote assistance, first-line workers can share their view with an expert, whilst staying hands-on to solve problems and complete tasks together.

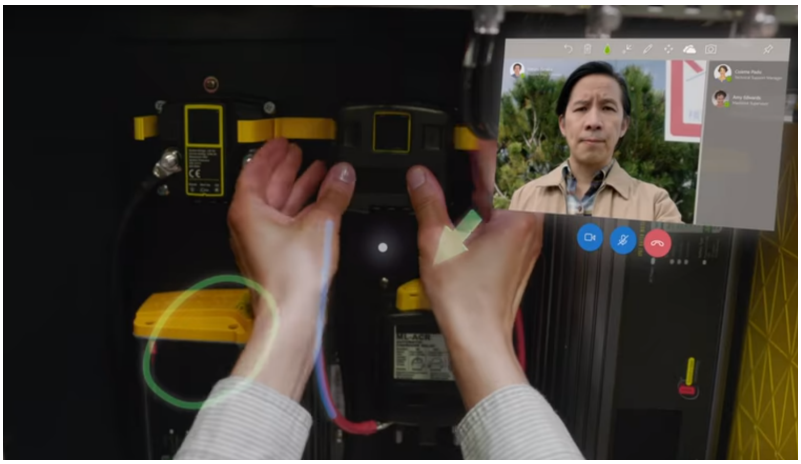


Figure 2.6: Sharing a view and solving a problem remotely with an expert [93].

Research works on collaboration in AR are widely presented in [85]. Authors also discuss the work in the domain of augmentation of face-to-face collaborative experiences and virtual co-location of people.

2.2.4. Entertainment

AR has been also applied to the entertainment domain by academics as well as commercial companies. AR technology provides a new type of games, which can attract players used to engaging in

virtual reality games. Bruce H. Thomas presented an AR gaming taxonomy, which classifies the different AR form factors taking into account display technologies (HMD, handheld, and spatially immersive displays) and whether the AR game is played indoor or outdoor [143]. This subsection provides examples of experiments that have been conducted based on AR games.

Zhou et al. built an experimental AR game to investigate the impact of 3D sound on task completion time and subjective feeling in the AR environment [169]. In this game, two users collaborate at the same time to rescue a virtual character. The game consists of three stages in which users perform searching tasks using aural cues. In the first task, players familiarize themselves with the AR interface, which enables to explore a virtual land augmented on the physical ground. In the second task, players fight together against a virtual enemy. Users need to transfer virtual bullets between each other to shoot the enemy. After defeating the enemy, users move to the third level. The goal of this stage is to find hidden princess using 3D sound cues. Experimental results of this study suggest that the use of 3D sound significantly improves task performance and accuracy of depth judgement. The results also indicate that 3D sound contributes to the feeling of human presence and collaboration and helps to identify spatial objects.

Avery et al. developed the *Sky Invaders 3D* game which was used to conduct a user study measuring enjoyment and intuitiveness of AR gaming compared to traditional PC-based game [21]. Authors implemented two versions of the game – an outdoor AR-based version and an indoor PC-based version – for the purpose of comparison. The goal of Sky Invaders 3D is to rescue the Earth from invasion of aliens by shooting down enemies. A player controls the AR-based version using an HMD. Conversely, in PC-based version of the game, the player uses a mouse and a keyboard to steer the game. Each participant taking part in the experiment played only one version of Sky Invaders 3D. Both versions recorded the scores and the time to complete the game. After playing the game, participants got a survey form consisting of 19 questions. Findings show that outdoor AR can improve the enjoyment of users. Furthermore, outdoor AR gaming was intuitive and easy to learn for beginner players. The survey results confirmed that significant differences exist in rating the enjoyment provided by the game between the outdoor AR version and PC-based version. Authors state that the outdoor AR factor was responsible for the improved enjoyment.

Figure 2.7: Playing the *It's a Pirate's Life* game [95].

Molyneaux and Benhimane developed an augmented reality game that uses real environment and reconstructs it into a dynamic game world [95]. The application runs on a prototype tablet in which the Intel RealSense camera system is embedded to provide RGB image-based camera pose and depth data (Fig. 2.7). The *It's a Pirate's Life* game has been designed to enhance the interactivity between a player and the real-world environment. The main goal of the game is to find treasures hidden within a computer-generated scene. The player acts as a pirate captain navigating a ship between a virtual sea

and real-world objects which become a part of the play scene. The player is responsible for controlling the wind by moving the tablet around the computer-generated scene to direct the virtual boat to hidden treasures.

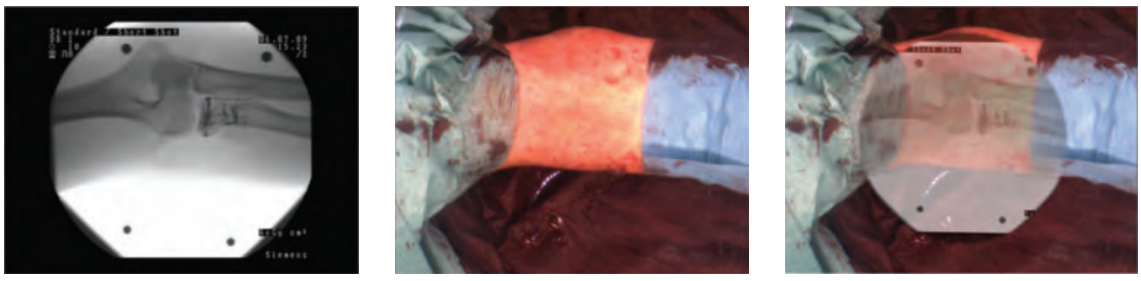
2.2.5. Medicine

AR is a very promising technology for medical applications. For instance, AR can improve the accuracy of surgical procedures, decrease the variability of surgical outcomes, lower trauma to the critical anatomical structures, and reduce radiation exposure [96]. Medical AR has been successfully applied to various domains of surgery such as neurosurgery, orthopedic surgery, and maxillofacial surgery. This subsection highlights examples of AR innovations employed in medicine.

Navab et el. [97] developed *CamC – Camera Augmented Mobile C-arm* that offers the AR technology for surgeons. A mobile C-arm is an X-ray device used in trauma and orthopedics surgery. *CamC* extends the C-arm device with a camera and a double mirror system allowing real time overlaying X-ray images onto optical images. The video camera is mounted in such a way that its optical center coincides with the C-arm's X-ray source. After a one-time calibration procedure, X-ray and optical images are co-recorded. The image overlay provides an intuitive interface for surgical guidance. Authors performed a series of experiments including assessment of technical accuracy of fusion of X-ray and optical images and measures of the X-ray radiation dose. Findings show that the presented technology is an intuitive and robust guidance solution for selected clinical routines. Moreover, the authors state that the use of the co-registered optical images can reduce the overall number of X-ray acquisitions and therefore the overall radiation dose to both patient and clinical staff is expected to be considerably decreased. Surgeons at the Leiden University Medical Center in Munich have performed more than 40 trauma and orthopedic procedures with the *CamC* system.

Blum et al. presented an extension of the CamC system with a brain computer interface (BCI) and a gaze-tracker device which are used to control X-ray AR visualization without using hands [27]. Authors conducted a pilot study to asses the potential of the technology. For the majority of the participants the use of BCI was intuitive. The additional use of gaze-tracking in a medical procedure was valuable with a potential to use in real surgical procedures.

Another medical AR system that guides surgeons during the needle insertion in radiofrequency ablation of the liver tumor was presented by De Paolis and Aloisio [38]. This system allows occlusion of virtual organs onto the patient's body. The solution gives a realistic impression that the synthetic organs are inside the patient's body. The system uses an optical tracker to track the position and orientation of the surgical instrument, which is then projected on a monitor. The position and orientation data are used to project the virtual surgical instrument. Moreover, the system visualizes the distance information



(a) C-arm X-ray image. (b) Optical video image. (c) *CamC image*.

Figure 2.8: Composing X-ray and optical images (a and b) into a *CamC* image (c).

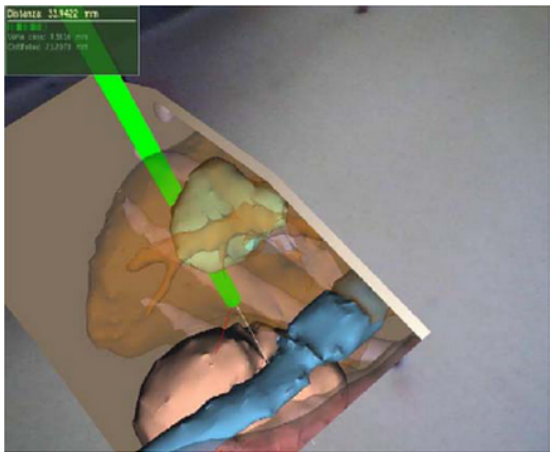


Figure 2.9: Visualisation of the needle insertion.

between the needle and the liver. When the distance between the surgical instrument and the organ is under a safety threshold, a video and audio feedback is provided to the medical staff. Authors reported that the system has been tested in the laboratory on a specific testbed.

2.2.6. Cultural heritage

AR is also an attractive technology for the cultural heritage domain. Many museums face the problem of limited museum's space and resources required to exhibit their whole cultural collections. Furthermore, some objects can be too fragile and in the end museum curators decide not to make them available to the public. Moreover, interactions between museums' visitors and artefacts are very restricted. It is not possible for a visitor to look at the cultural objects from all angles or to compare them and study in different contexts [162]. AR can offer a great help to solve the above-mentioned difficulties. AR provides solutions enabling visualization of 3D models representing cultural heritage collections. Also, visitors can interact and experience exhibitions in a new way.

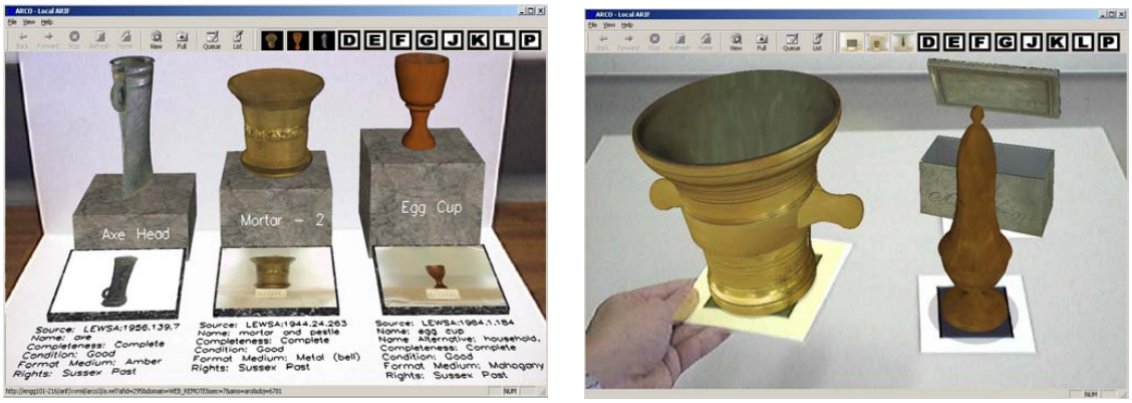


Figure 2.10: Virtual exhibitions displayed in ARCO AR interface [156].

Walczak et al. proposed the *ARCO – Augmented Representation of Cultural Objects* – system that enables museums to build and manage virtual and augmented reality exhibitions based on 3D models of artefacts [156]. The system consists of three architectural components which help to produce, manage, and visualize virtual exhibitions. An administrator of virtual exhibitions can decide which objects, how and where should be published. For instance, a virtual exhibition can be visualized in three ways – using Web, VR, and AR interfaces. Moreover, *ARCO* allows to build interactive learning scenarios, in which

a user is not a passive visitor, but an active actor and player. Using a tabletop device, an end-user can examine and learn about the cultural objects by manipulating digital artefacts in a context of real objects. Figure 2.10 presents examples of two virtual exhibitions created with the ARCO system.

Authors reported that after initial demonstrations of the ARCO system, museum visitors provided positive feedback, proving that AR interface is a very good supplement to much more mature presentation methods, such as Web and VR.

Rumiński and Walczak developed *MARAT – Mobile Augmented Reality Authoring Tool* which is an easy-to-use mobile authoring application for AR presentations. The application extends the ARCO system [119]. One of the key requirements for the application was implementation of an authoring method that is easy to use for a museum creator and does not require programming skills. The application is intended also for use by end-users, e.g., exhibition visitors, to access the AR content. Typically, users do not have the rights to create or modify AR presentations. These users run MARAT in the read-only mode without access to the functionality that enables modification of the AR content.

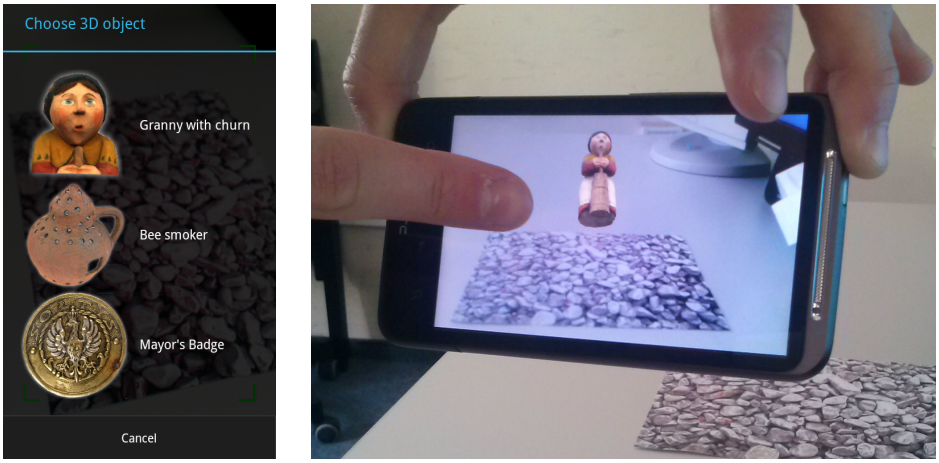


Figure 2.11: Assignment of objects to an image marker using mobile device [119].

A content designer, who is responsible for creating and managing virtual exhibitions, uses the mobile AR application to assign virtual objects to real locations (image markers) and to set their presentation and interaction properties. MARAT enables setting object presentation and interaction properties in a user-friendly interactive WYSIWYG mode. Both the assignments and the properties of objects are stored on the ARCO server and therefore are visible for every user of the MARAT application.

Visitors, equipped with mobile devices with the MARAT application installed, can browse a virtual exhibition. They can view the virtual exhibition through mobile devices equipped with cameras and can interact with virtual objects.

2.2.7. Automotive

AR brings many advantages to the car industry – in particular – in improving drivers' safety and navigation [35, 74]. A head-up display (HUD), that does not distract the driver while driving, was employed in Mercedes-Benz cars [35] (Fig. 2.12). The AR system overlays a digital read-out of navigation instruction, driving speed, and cruise-control settings on the windscreen.

While the AR solutions implemented in the premium car segment are expensive, small companies work on cheaper AR alternatives. One of the notable projects is *Hudway Glass* [74] funded by the Kickstarter crowdfunding platform [73]. *Hudway* delivers simple, versatile, and affordable HUD software and glass-based accessory that helps a user to safely and comfortably drive a car. The idea of

Figure 2.12: A head-up display for Mercedes-Bentz cars [35].

Hudway is to project navigation information onto a special glass that mirrors the view of the smartphone's display. This combination eliminates many issues associated with projecting images onto the windshield, e.g., doubling of the image or lack of reflection during daytime.

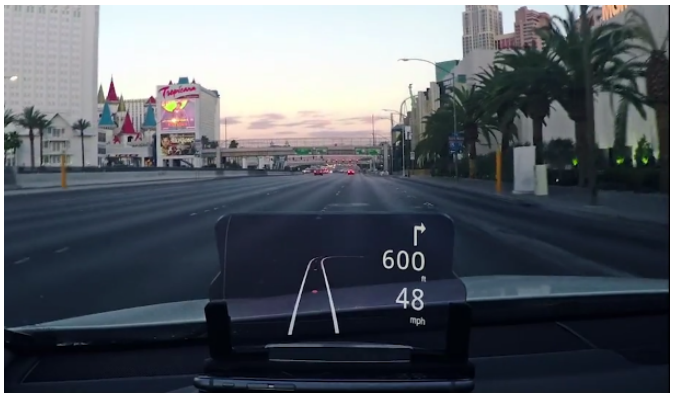


Figure 2.13: Hudway Glass in use [74].

AR can be used not only for improving drivers' safety and navigation, but also in maintaining procedures for auto mechanics [133]. Stanimirovic et al. initiated the project called *MARTA – Mobile Augmented Reality Technical Assistance* in order to deliver technical assistance instructions, which guide a technician step-by-step through maintenance tasks. The *MARTA* application provides robust image-based tracking of specular vehicle surfaces and overlaying of interactive instructions in the camera view.

2.2.8. 3D content design

Researchers demonstrated that AR technology can be used for 3D design and production of 3D synthetic content. Krichenbauer et al. built an AR user interface in the form of a plug-in for Autodesk Maya [76]. The system was designed on the basis of results of the conducted survey aimed at 3D media professionals responsible for 3D computer graphics content creation [77]. Based on the findings, authors identified requirements that are needed for professionals while designing 3D models with AR UIs. The following list presents professionals' needs:

- ergonomic design;
- collaboration support;
- high amount of features;

Figure 2.14: Guiding an auto mechanic with the *MARTA* application [133].

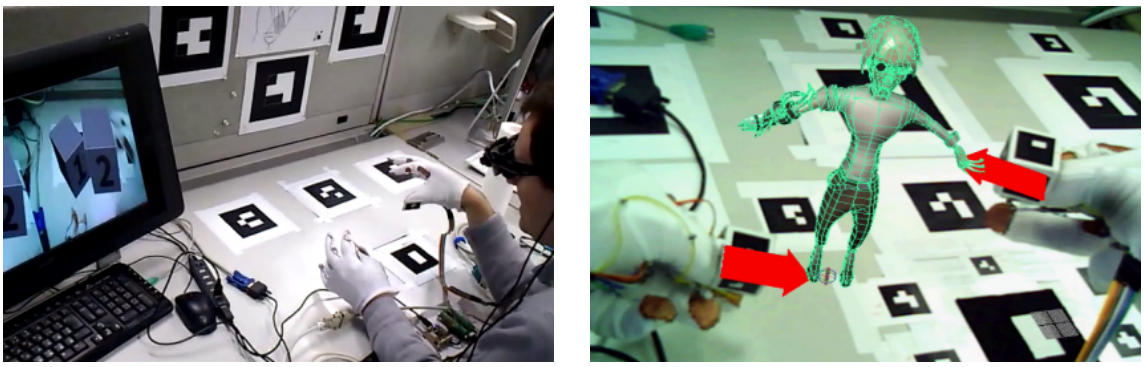


Figure 2.15: Editing a 3D object with an Augmented Reality User Interface [76].

- fast and intuitive navigation;
- support for 2D and alphanumeric operations;
- increase in productivity.

Authors implemented the gathered requirements in an AR system consisting of a head-mounted display and thin cotton gloves with conductive materials sewed on them, as presented in Fig. 2.15, allowing to interact with virtual 3D models in the work area. To work with the system, a user needs to wear the HMD device and tracked gloves which turn an empty work area into an augmented reality workspace. Next, primitive virtual content appears on the desk and can be edited directly with hands. While editing 3D content, additional observers are able to see a video stream on a projector, and thus, share the user's view.

Steptoe developed a stereo camera rig to provide immersive video see-trough augmented reality for the Oculus Rift, calling it AR-Rift [135]. The author mounted two web cameras on the Oculus Rift device to capture world from the perspective of the user's eyes (as presented in Fig. 2.16). Additionally, to track head and hands in 3D, the author connected a motion capture system to the Oculus Rift. Such combination of devices can deliver a highly immersive user experience – not only in VR, but also in AR.

To enable interaction with synthetic content, the author implemented a novel 3D user interface which features a panel attached to one hand marker and a manipulator in the position of the fingertips on the other hand.

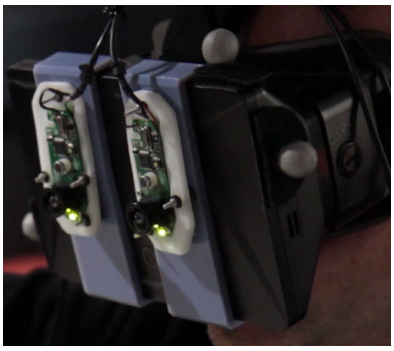


Figure 2.16: AR-Rift developed by Steptoe [135].

With the solution proposed by Steptoe, a user is able to create primitive 3D models, dock virtual objects at a specific position, place virtual displays on the walls of the real room, delete virtual objects, while wearing a virtual Oculus Rift to enable transition between AR and VR.

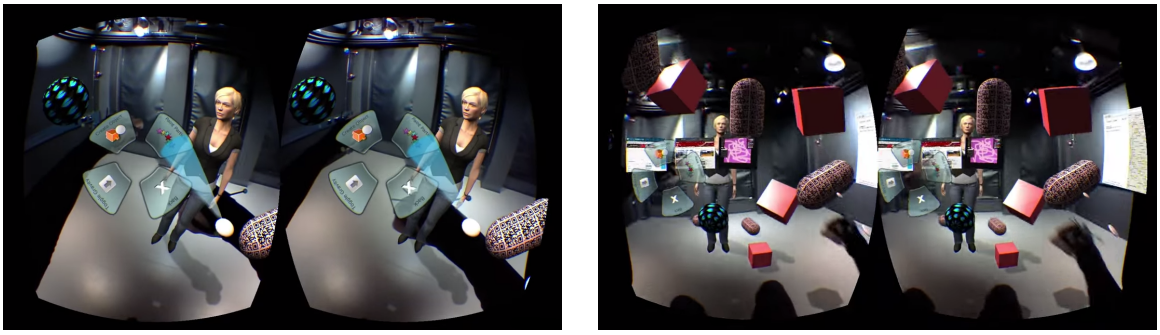


Figure 2.17: Creating virtual objects using AR-Rift [135].

Piumsomboon et al. demonstrated a novel AR system – *Gesture-Speech Interface for Augmented Reality* (G-SIAR) – where a user is able to interact with synthetic content through a combination of direct and indirect natural interaction techniques in an AR environment [107]. Hand tracking and speech recognition technologies were used to design high precision free-hand gesture and speech interface input. G-SIAR provides both direct manipulation technique using free-hand gesture called *Grasp-Shell* (G-Shell) and indirect multimodal technique called *Gesture-Speech* (G-Speech). A comparison of these techniques is extensively described in [106]. G-SIAR consists of two wide-angle stereo cameras mounted on the Oculus Rift display providing an immersive user experience across the mixed reality spectrum. For instance, users can transit between AR and AV environments using hand gesture, where their real hands are visible within a virtual environment.

On the basis of G-SIAR, authors implemented a 3D design sandbox application which assists in dynamic creation of virtual objects allowing users to design their own content. To create virtual objects users need to use G-Shell and G-Speech techniques. For instance, a user can draw the outline of an object by using his/her finger, to bring into virtual existence a solid model. Figure 2.18 presents a user who creates virtual objects using his hand with one of the G-Shell gestures. The AR system supports importing of external models in various formats. Moreover, an export of created objects is also possible while developing 3D content.

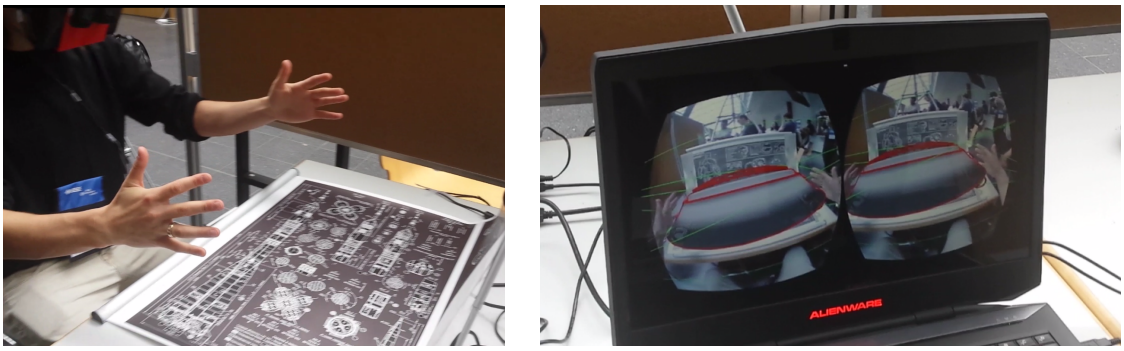


Figure 2.18: Interacting with just created object using *G-SIAR* [107].

2.2.9. Architecture and real estate

Researchers and commercial companies applied augmented reality to visualize content and data. For instance, Lee et. al presented the *CityViewAR* application to provide geographical information about destroyed buildings and historical sites of the city of Christchurch, which were damaged by several major earthquakes in 2010 and 2011 [83]. Furthermore, *CityViewAR* allows to visualize virtual 3D models of buildings on the real site where – before earthquakes – they were situated. Figure 2.19 depicts visualization of a virtual building on a view of real-world building affected by the earthquake. On the basis of the *CityViewAR* application, authors conducted an experiment showing that the AR interface enhanced the user experience while exploring the city of Christchurch using the application.

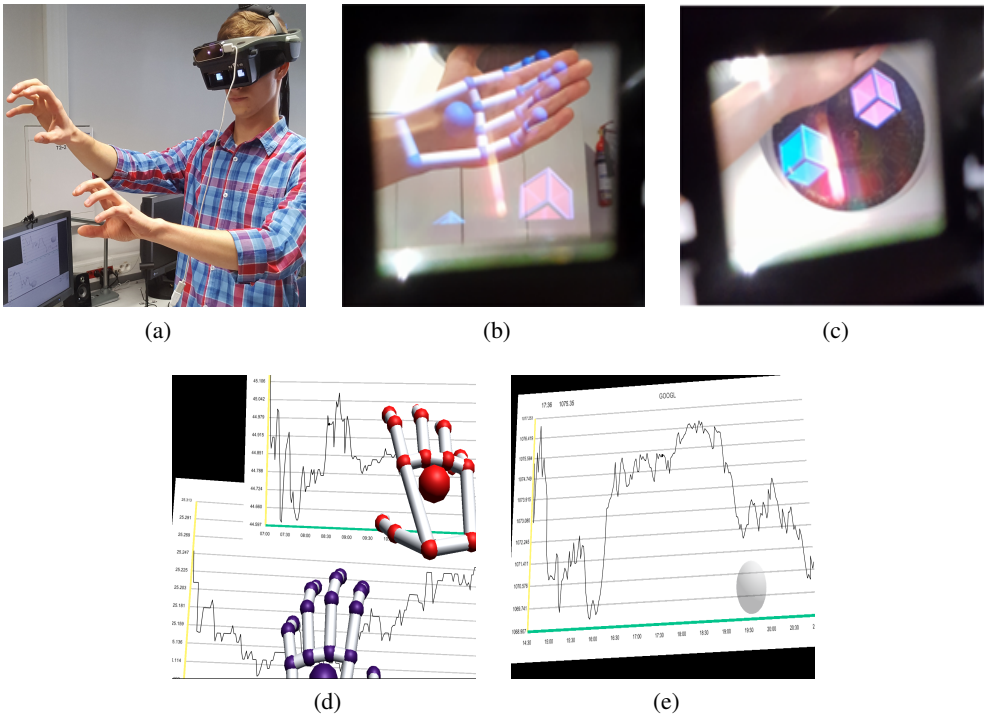
Figure 2.19: Presenting virtual building on a view of destroyed place after the earthquake [83].

The real estate sector is another example where Augmented Reality technology is increasingly being used providing benefits for property sellers, agents, and buyers. Commercial companies developed numerous AR applications (e.g., Lux AR and Zoopla) responsible for projecting information about the properties that are available for sale over the view of the houses [13, 84]. The overlaid data such us price, direction, and distance from the user's location are visible to the user when he or she points a mobile device's camera toward houses. On the one hand, this kind of AR applications provide a great tool for property sellers. On the other hand, users reduce time that must be spent dealing with the real estate agent. Moreover, these applications give an innovative experience to potential buyers, while they are searching for new homes.

Apart from searching information about homes, in some cases, buyers are able to see how the interior of not-yet-finished houses will look after completing the construction process, just by viewing virtual model of the place using a mobile device [109]. Prospective buyers can check out how their homes will look when decorated in different ways, making the house more appealing to the buyers.

2.2.10. Visualization of data

Rumiński et al. presented a novel mixed reality system for supporting stock market trading [117]. The system is designed to enhance traders' working environment by displaying an array of virtual screens visualizing financial stock data and related news feeds within the user's surroundings (Fig. 2.20). The authors combined the nVisor ST50 headset with InteriaCube4 and Leap Motion devices to enable tracking of head orientation and controlling the VR/AR environment with hands. With the use of this AR system, end users can create and control the virtual screens displaying stocks data directly using their hands in 3D space.



(a) (b) (c)

(d) (e)

Figure 2.20: Creating, configuring, and visualizing virtual stock charts [117].

2.3. Summary

This chapter has introduced a relatively new field of computer science – augmented reality. The first AR display device and the application developed by Ivan Sutherland has been described. Next, two definitions of AR, provided by Milgram et al. and Azuma, were presented. Afterwards, the diversity of AR domains and examples of various AR applications have been discussed. The introduced AR systems share common features – AR functionality is fragmented between independent programs and each one has to be individually installed and experienced by end users. Moreover, content and data are "closed" within a single application and cannot be reused in others scenarios/applications.

3. Methods of Modeling of Augmented Reality

In this chapter, AR development tools for creating augmented reality applications are discussed – from low-level software libraries and frameworks requiring programming skills to use them, through simple easy-to-use authoring tools for non-technical users, to AR declarative languages used for designing AR applications. Also, this chapter explores the role of semantic web and how it was applied to AR. The chapter introduces semantic web standards and popular serialization formats with examples. After that, applications of the semantic web technology in augmented reality systems is presented. Next, the limitations of existing methods of modeling AR environments are discussed. Finally, a summary of the chapter is provided.

3.1. AR software libraries and frameworks

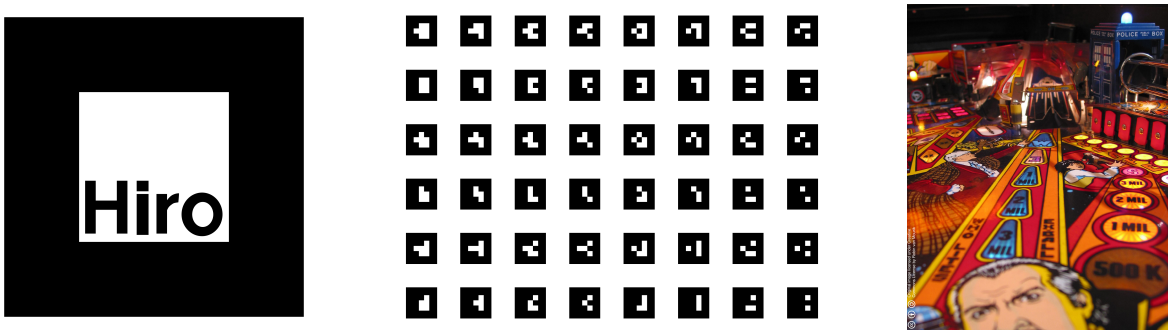
3.1.1. Overview of low-level AR development tools

Augmented reality development tools have diverse target application platforms. Some tools can be used for developing desktop AR applications, while others are designed for mobile AR programs. Due to remarkable progress in mobile hardware performance, widespread use of mobile AR technology is now possible. This section is focused on presentation of the state of the art of AR development tools for mobile AR applications.

Typically, software libraries and frameworks for developing AR applications provide functionality responsible for tracking real objects and displaing virtual ones. One of the well-known AR software libraries, primarily developed by Kato and Billinghurst, is ARToolKit [18, 70, 71]. ARToolKit was first demonstrated publicity at the ACM SIGGRAPH conference in 1999 [25]. Since then, numerous desktop-based AR applications have been created with the use of ARToolKit. The library has been ported to mobile platforms and is still further developed by a commercial company – DAQRI [36].

ARToolKit calculates the real camera position and orientation relative to a fiducial marker, characterized by a black outlined square with a pattern inside, allowing to overlay virtual objects. Currently, it supports classical black square markers (Fig 3.1a), multimarkers (Fig 3.1b), and natural feature tracking (Fig 3.1c). The library is multi-platform, running on Windows, Linux, Mac OS X, iOS and Android operating systems. Developers can also integrate the library with the Unity3D game engine [139] using the ARToolKit plug-in. ARToolKit supports both video and optical see-through devices. ARToolKit provides an open source license for marker-based applications and a commercial version of the software based on natural feature tracking of arbitrary images.

Before DAQRI invested in development of application programming interfaces (APIs) covering modern mobile platforms, developers could use NyARToolkit [9] designed for Java/C#/Android, AndAR available only for Android [41], and CoreAR [5] for Objective C built by developers connected with the open source community. However, these forks are not further developed and it is recommended to use the newest ARToolKit's code.



(a) Fiducial marker. (b) Multimarker. (c) Image-based marker.

Figure 3.1: ARToolkit's types of tracking markers [18].

The next open source framework for building augmented reality applications is DroidAR [63]. This framework uses location and vision-based marker tracking methods. DroidAR supports only the Android platform. The software is based on the OpenCV computer vision library [68] and uses square black markers for tracking – similarly to ARToolKit. DroidAR is available under commercial license and can be freely downloaded and used for non-commercial use under the GNU GPL v3 license [49]. Additionally, DroidAR provides a testing tool for desktop computers, allowing very fast development cycles. Unfortunately, the framework is not well documented and a developer can only rely on screencasts posted on YouTube and some comments included within the code.

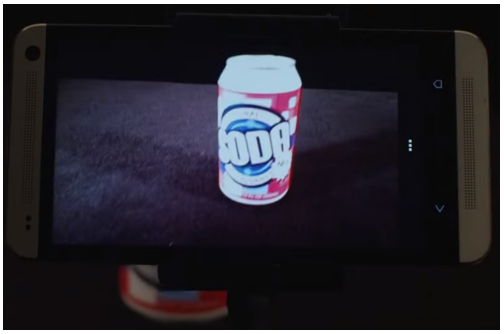
BeyondAR is an open source framework that provides location-based tracking for AR applications [125]. The framework supports only the Android platform. Authors have provided the source code of an AR game based on the framework. To recognize a location of a virtual object, the BeyondAR framework uses device's sensors such as GPS, accelerometer, and compass. BeyondAR is well documented in comparison with the DroidAR framework. The AR applications built on the basis of BeyondAR can be freely distributed under the Apache license.

To develop AR location-based applications for the Android platform, developers can also use the HIT Lab NZ OutdoorAR framework [82]. The framework consists of a software library, a web-based server, and a web-based authoring tool. OutdoorAR is designed with a component-based approach allowing developers to reuse software components responsible for data storage, tracking sensors, network communication, 3D graphics and sound rendering, as well as tools for managing UI elements in order to build outdoor AR applications. The framework allows developers to focus on high-level design and development of the domain content, logic, and user interface. It also provides direct access to low-level functionality.

For the iOS platform, developers can use a commercial framework called PanicAR that enables development of location-based AR applications [52]. PanicAR renders points of interest (POIs) on a view of the real world. The framework provides a radar tool drawing POIs on a map that can be easily integrated with an application. The framework requires an API-key to unlock all functionality. As long as no valid API key is used during initialization, the application displays watermarks in the AR view and on POI labels. The authors published also a beta version of the framework for Android, but without any documentation and tutorials.

Another interesting framework designed for the iOS platform is 3DAR. 3DAR is an AR toolkit developed by Spot Metrix, Inc. [92]. It is based on GPS and IMU sensors. Authors stated that in most cases, the best approach is to first make 2D map annotations. After that a developer needs to implement an extension in 3D view of the map. The 3DAR SDK is freely available for use in any iOS application

(a) An image-based multimarker composed into a box.



(b) Tracking an image wrapped onto a can.

(c) A frame marker.

(d) Real-world object as a marker.

Figure 3.2: Vuforia's types of tracking markers [110].

provided that the logo of the Spot Metrix company is visible on at least one AR view. A developer can also use commercial license to develop location-based applications without any limitations.

Mixare is an open source augmented reality engine for geolocation applications covering both iOS and Android platforms [53]. The framework is published under the GNU GPL v3 license. It works both as an autonomous application and as a library for development of other applications. The authors published a WordPress plugin to easily create points of interest using a map on any WordPress-powered website. With it, a user can add POIs with a few simple steps and choose whether POIs should be publicly visible or available only for the user. Every set of POIs is displayed on a website with a QRCode that can be scanned. Scanning the barcode will automatically launch the Mixare browser on the device (if it is installed).

To cover both iOS and Android platforms, while developing an AR application, developers can use the Kudan AR engine. The framework provides marker-based and markerless tracking, as well as various forms of 3D rendering [44]. The solution uses a gyroscope as a sensor to control markerless tracking techniques. The framework supports FBX [20], OBJ [140], and COLLADA [60] 3D content formats. In order to develop an AR application, a developer can use free license keys for Kudan's components. To use the engine in commercial applications, a paid license per application is required.

With Aurasma Technology, provided by the HP company, developers are also able to implement AR applications on both iOS and Android platforms [3]. The SDK was created for digital agencies responsible for launching and managing AR campaigns. The Aurasma SDK consists of a static library (AurasmaKit), a sample application which embeds the Aurasma SDK (AKTest), and a set of images used for tracking by the AKTest application. Aurasma became very popular due to the simplicity of tools through which users can create their own AR experiences – so called Auras – and share them to the Aurasma's community.

ARLab company is another provider of mobile AR tools [130]. ARLab's SDK consists of five products through which developers can create AR applications on iOS and Android platforms. The SDK allows developers to build applications based on GPS, IMU sensors as well as a natural feature tracking method. Although the SDK is well documented, the community of developers is relatively small. Developers are forced to collect virtual points to get tutorials, watch videos, and get code examples. To get these points a developer has to buy a concrete product.

One of the most popular augmented reality software development kits, which enables building mobile AR applications, is Vuforia [110]. Originally, Vuforia was developed by Qualcomm Inc., but on November 3^{rd} 2015 it was sold to PTC Inc. Vuforia provides computer vision based image recognition methods that are able to recognize and track arbitrary images, objects, text, fiducial markers, and reconstruct environments. It allows to write native applications for Android and iOS platforms. With Vuforia it is not only possible to create augmented reality applications for mobile devices, e.g., smartphones/tablets, but it is also possible to build VR and AR applications for video and optical see-through devices. Vuforia recognition and tracking capabilities can be used on a variety of images and real objects, such as: flat images (as presented in Fig. 3.1c); multiple flat images that can be arranged into regular geometric shapes, e.g. boxes (Fig. 3.2a); images wrapped onto cylindrical objects, e.g., cans, bottles (Fig. 3.2b); 512 numerically encoded small markers that can be used with any image – so-called frame markers (Fig. 3.2c); real-world objects, e.g. toys (Fig. 3.2c). Moreover, Vuforia SDK is capable to recognize text from a dictionary of ~100,000 English words.

The Vuforia SDK is one of the three components constituting the full Vuforia platform. The second component consists of three tools, which help to create databases of recognizable objects, scan 3D objects into a format that is compatible with the Vuforia client-side library, and assist with the calibration for optical see-through eyewear devices. The third component is so-called Cloud Recognition Service and helps to recognize a large set of images on-line. This component can be used to automate systems workflows by direct integration into a content management systems.

The next notable SDK for AR development is Wikitude SDK provided by Austrian-based Wikitude GmbH company [10]. The company is mostly known from the location-based AR application with the same name ("Wikitude"). The Wikitude SDK supports location-based and image-based recognition and tracking methods. To build AR applications, a developer may use web technologies (HTML, JavaScript, and CSS), which enable developing cross-platform augmented reality experiences. With Wikitude SDK it is also possible to write native code for Android and iOS platforms. The Wikitude SDK consists of computer vision engine library, the JavaScript library allowing to program AR experiences nested within HTML code, native application programming interfaces (APIs). The Wikitude platform provides cloud-based tools, such as: Wikitude Cloud Recognition allowing to visually search image targets; Wikitude Targets API for developing own CMSs allowing to create and manage large target image collections; and the authoring application Wikitude Studio. Wikitude supports not only smartphones and tablets, but also smart glasses devices, such as: Google Glass, Epson Moverio, Vuzix, and Optinvent.

Another well known augmented reality technology is Layar founded by a Dutch company, which is a part of the Blippar Group. Layar gained international attention as one of the first mobile augmented reality browsers. Layar, which is a competitor for Wikitude, equips developers with a set of tools enabling them to build augmented reality applications that can run on Android and iOS platforms. The Layar SDK is a static library providing location-based and vision-based methods that help to create various AR presentations – so called "layers". Similarly to the Wikitude platform, Layar consists of REST-based services integrating external CMS applications and an easy to use authoring application, called Layar Creator. Layar does not support smart glasses devices.

It is also worth to mention a German-based company – Metaio – which provided AR technology for mobile devices. Metaio SDK is a static library that supports location-based and vision-based (including fiducial markers and natural feature tracking) AR applications. Similarly to Vuforia, Metaio SDK provided recognition and tracking capabilities for rigid real-world objects. Currently, Metaio products are no longer available for purchase after the Apple company had acquired Metaio.

Finally, in addition to the presented augmented reality software libraries and frameworks, there are other SDKs that can be used for developing desktop AR applications on Windows, Linux and MacOS systems. The Institute for Computer Graphics and Vision at the Technical University of Graz provides a list of all published software [112]. It is also worth to mention that many listed libraries have not stood the test of time and the provided links lead currently to websites that do not exist anymore.

3.1.2. Comparison of AR software libraries and frameworks

Table 3.1 presents a comparison of augmented reality software libraries and frameworks that can be used for developing mobile AR applications. Each library/framework is assessed against five general evaluation criteria. Solutions that are not available to use (e.g., Metaio) or forks based on ARToolkit such as NyARToolkit [9], AndAR [41], and CoreAR [5] are not considered.

The first criterion are the platforms supported by the development environment. The vast majority of presented toolkits support Android platform (92%) – except the 3DAR. The majority allow also to develop AR applications on the iOS platform (77%). Only ARToolKit supports alternative operating systems including mobile Windows Phone and desktop-based platforms, such as Windows, Mac OS, and Linux.

The second criterion concerns the license under which the AR software is available, i.e., free or commercial. The majority (85%) of evaluated AR software is available under the commercial license. Only BeyondAR is available under the Apache License version 2.0 (ASF) [48] and Mixare under the GNU GPL, version 3 – which means that any software built on BeyondAR or Mixare can be freely used, distributed, modified, under the terms of the license, without concern for royalties. The slight majority (54%) provides solutions under proprietary licenses that are free of charge (except Aurasma and Layar), but, in most cases offering more advanced paid options.

The third criterion represents availability of the documentation and support for the presented developments tools – namely, whether the documentation is up-to-date, provides a 'Getting started' section, tutorials, and a community forum. Helpful 'Getting started' documentation and rich tutorials with code examples are provided for all the compared software. However, for most of open source projects (except ARToolKit) the documentation is out of date. Furthermore, PanicAR provides only documentation for the iOS version of the framework. Only 46% of evaluated solutions made the community forum available for the developers. ARToolKit, Vuforia, and Wikitude have passed all 'Documentation and support' criteria.

The next criterion indicates what kind of tracking is used by the software. Four tracking methods have been examined – based on detection and tracking of: markers, images (NFT), sensors, and real-world objects. The results are the following: 38% use marker-based method; 54% provide image-based techniques; 85% use sensors, such us: GPS, accelerometer, gyroscope, and magnetometer; and only Vuforia is capable of tracking real-world objects.

Finally, the support for wearable devices is reported. Only five solutions provide tools for developing AR applications for wearable devices. BeyondAR and Layar support Google Glass. ARToolKit, Vuforia,

and Wikitude allow developers to build AR applications that are able to render stereoscopically with an optical see-through display, providing a different perspective for each eye.

To summarize, regardless of the license, ARToolKit, Vuforia, Wikitude provide the richest set of tools covering most of modern mobile platforms and devices. These solutions can be considered to use for building contextual augmented reality environments.

3.2. Integrated AR design environments

3.2.1. Visual designing of AR

The technologies described in Section 3.1.1 require considerable programming skills from developers. Also, developing AR applications with low-level SDKs is typically a time-consuming process. In some cases it can be convenient and helpful to rapidly prototype an AR application to present an idea to the end-users or clients before undertaking a major development process. Within this subsection, an overview of various easy-to-use AR tools, used for visual and rapid prototyping of mobile AR applications, is presented.

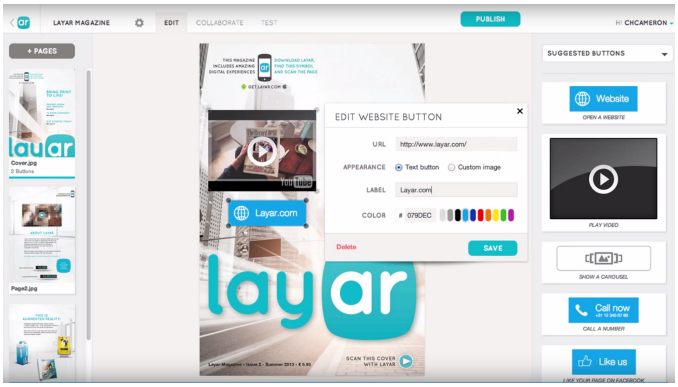


Figure 3.3: A web-based drag and drop interface of Layar Creator [8].

The Layar Creator is an easy-to-use web application providing high-level functions based on the Layar library presented in previous section. The application helps to design AR presentations by allowing to place images, videos, texts, and buttons onto markers [8]. A user only needs to upload to a server an image or a PDF file that will serve as a marker for AR content. A user can drag and drop these elements on a marker image and adjust the position and orientation of the displayed AR content. After the AR presentation is finalized it can be published by pressing the publish button. Then, the AR content can be accessed by aiming the Layar AR browser application on a marker image. Figure 3.3 depicts an example view of the Layar Creator's interface.

The next web-based augmented reality tool enabling users to create, manage, and publish AR presentations is Wikitude Studio [11]. The application is appropriate for non-technical users, which are not required to have programming skills. A simple easy-to-use drag-and-drop interface (Fig. 3.4) gives capabilities to augment a view of real world objects with text, images, HTML widgets, video, and 3D objects. After positioning a content element on a view of real world object, a developer can preview the created AR presentation. After that, the AR presentation can be described with using tags and description and then published to Wikitude Cloud to make it available in the Wikitude browser. A developer can also export the AR presentation to his/her own application.

Table 3.1: A comparison of low-level AR software libraries and frameworks for building mobile AR applications.

	3DAR	ARLab	ARToolKit	Aurasma	BeyondAR	DroidAR	Kudan	Layar	Mixare	OutdoorAR	PanicAR	Vuforia	Wikitude
Platform													
Android	✗	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓ -	✓	✓
iOS	✓	✓	✓	✓	✗	✗	✓	✓	✓	✗	✓	✓	✓
Others	✗	✗	✓	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
License													
Open Source	✗	✗	✓	✗	✓	✓	✗	✗	✓	✓	✗	✗	✗
Free	✓	✗	✓	✗	✗	✓	✓	✗	✗	✗	✓	✓	✓
Commercial	✓	✓	✓	✓	✗	✓	✓	✓	✗	✓	✓	✓	✓
Documentation and support													
Doc. is up-to-date	✗	✗	✓	✓	✗	✗	✓	✓	✗	✗	✗	✓	✓
Getting started	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Tutorials	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓
Community forum	✗	✓	✓	✗	✓	✗	✗	✗	✗	✓	✗	✓	✓
Tracking													
Marker	✗	✓	✓	✗	✗	✓	✓	✗	✗	✗	✗	✓	✗
NFT	✗	✓	✓	✓	✗	✗	✓	✗	✗	✗	✗	✓	✓
GPS, IMU Sensors	✓	✓	✗	✓	✓	✓	✓ ×	✓	✓	✓	✓	✗	✓
Real-world object	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗
Support for wearable devices	✓	✗	✓	✗	✓	✗	✗	✓	✗	✗	✗	✓	✓

- beta version available
× only a gyroscope is used

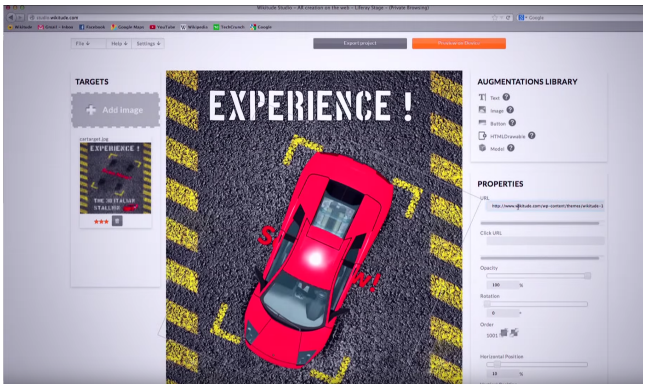


Figure 3.4: A web-based drag and drop interface of Wikitude Studio [11].

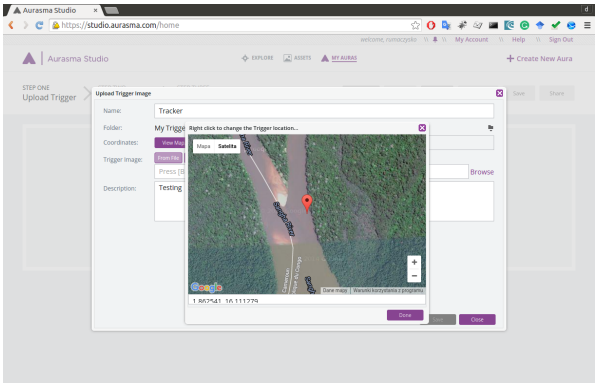


Figure 3.5: A web-based drag and drop interface of Aurasma Studio [4].

Another example of web-based AR authoring tools for mobile devices is Aurasma Studio. This application, similarly to Wikitude Studio, provides a possibility to augment images when using a smartphone or a tablet [4]. Creation of an AR presentation is very simple and a user is not required to have programming skills. First, the user needs to upload a so-called Trigger Image to the server. This image should be described by the user. This image will be used to access digital content. Next, the user selects a digital content object from a library, e.g., an image or a video. Further, the user needs to position the digital content by adjusting and rotating the content object until it has the most appropriate position. Finally, after saving the work, a so-called 'Aura' is available for use. The user can make his/hers 'Aura' public. Thus, every user can access it using the mobile version of Aurasma application.

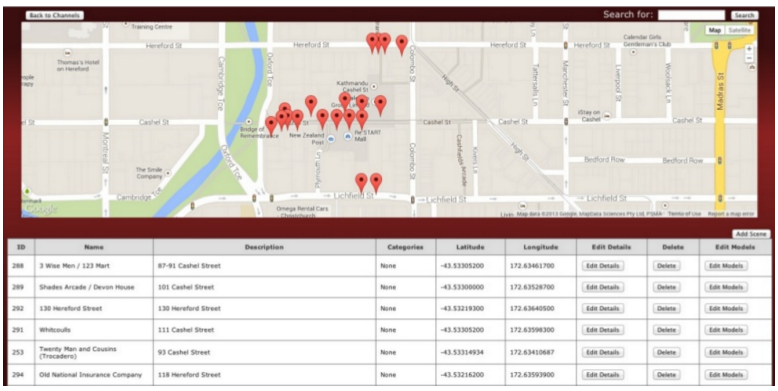


Figure 3.6: A web-based drag and drop interface of the OutdooAR web application [82].

Creation of location-based mobile AR applications is also possible with the use of web-based interfaces. For instance, one of the crucial parts of the OutdoorAR framework is an authoring web-based application where a user can browse, modify, and manage geo-located scenes' information [82]. With this tool a user is also able to create new AR scenes by describing the details of a new scene in a web form, uploading related media assets, and placing them on a map. Finally, these media data can be retrieved by a mobile application that implements the OutdoorAR framework. Another interesting geo-tagging web-based authoring tool is Hoppala Augmentation. The user interface of this application is based on a map view component, in which a user provides geo-tagged content. These data created by the user can be accessed on a number of mobile AR browsers such as Wikitude, Layar, and Junaio. Finally, Aurasma Studio provides optional functionality to describe an image with GPS coordinates – simply by dropping a point on a map.

There is a number of desktop-based AR authoring tools available, which are capable to publish virtual objects onto mobile platforms. For instance, the *ComposAR-Mobile* application is a cross-platform content authoring tool for PCs and mobile phones [159]. The application provides a GUI-based interface that simulates a mobile phone allowing quick prototyping of AR applications on a desktop computer. The application enables a designer to select markers and 3D models stored on the local system and to link virtual objects with markers. Once a marker and corresponding 3D models are linked, a user is able to change the position, rotation and scale of the assigned 3D models in the AR scene. Keypad based interaction within the AR scene can be simulated using a virtual keypad in the visual keypad component. One of the advantages of this solution is the integration with Python code scripts that are interpreted, so that immediate feedback "on the fly" can be seen from the AR scene simulator. In the end, the application produces an XML description of the AR scene that is further processed by a mobile application.

Another example, which has similar functionality to ComposAR-Mobile, is the D'Fusion Studio application [138]. This tool also provides a mechanism for prototyping virtual objects on a desktop computer and publishing AR presentations onto mobile platforms. However, in contrast to ComposAR-Mobile, D'Fusion Studio covers majority of modern operating systems, such as: Windows, Mac OS, Linux, iOS, and Android. Moreover, D'Fusion Studio allows the designer to develop more sophisticated interactive scenarios with the LUA scripting language. In addition, D'Fusion Studio provides face tracking capabilities. Figure 3.7 presents an example view of the user interface of this application.

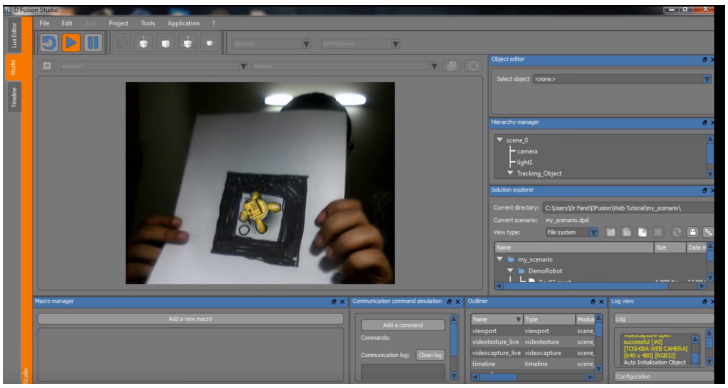


Figure 3.7: A desktop-based interface of D'Fusion Studio [138].

Finally, a user can quickly prototype AR ideas with the use of AR browser applications such as Layar, Aurasma or Wikitude. These are free to use mobile applications available on digital distribution platforms, such as Google Play [6] and Apple App Store [1], that connect back to servers for providing

virtual objects. Creation of an AR presentation is very simple and the user is not required to have programming skills when using these tools. Each of these application is base on a similar use case. For instance, with the Aurasma application the user needs to capture an image representing a view of a real world object with the mobile device. Next, digital content from a library, e.g., an image or a video sequence should be selected and associated with the captured image. This image will be used to access the digital content. The last step is to position the content object by adjusting and rotating until it has the most appropriate fit. After saving a work, the so-called 'Aura' is available for use. The user can make the created 'Aura' public – similarly as in the case of Aurasma Studio when developing AR presentations.

3.2.2. AR programming languages

To simplify the process of building augmented reality applications, content designers can use high-level declarative programming languages. These languages require content designers to specify what and where should be augmented, what should be displayed on a view of real-world objects, and what can be the interactions between users and content. In most cases, users do not have to have specific technical knowledge to use non-procedural languages to model AR presentations. One of the biggest advantages of using declarative languages is simplified development process, which does not require code compilation. Moreover, the process of modeling is relatively easy to do in comparison to starting up a project from scratch using specific low-level software libraries.

Several research works have been performed in the domain of declarative languages for building AR environments. For instance, Ledermann and Schmalstieg developed an XML-based language for authoring AR presentations [81]. The Augmented Presentation and Interaction Language (APRIL) uses UML state charts to design the flow of AR presentations. A content designer creates state charts, which are next exported as files in the XML Metadata Interchange (XMI) format [104] – the standard format for serializing UML diagrams. The XMI files are translated into the APRIL format and included in the overall presentation description. This approach allows to use any UML editing tool capable of exporting to XMI format. Finally, each AR application encoded in the APRIL format is transformed into the representation of the Studierstube format [126].

The APRIL language allows the programmers and content designers to specify four top-level elements – `setup`, `cast`, `story`, and `behaviors` – which enable customization of AR presentations. The setup element describes hardware setup on which the application will run. Furthermore, this element separates the description of content and behavior of the presentations from all aspects that depend on the hardware. APRIL allows to store descriptions of various hardware definitions in separate files, and run a single application on different hardware setups. This approach allows also to share and reuse a single description of hardware setup among multiple AR presentations without the need of changing content. The next element of APRIL language – `story` – is an explicit definition of a temporal structure of AR presentations. This element is composed of individual scenes, in which a predefined sequence of `behaviors` is executed by `actors` grouped in a `cast` element. An `actor` element may contain a virtual object, a video clip or a sound. Actors can be nested in other actors, so a single actor can represent a group of other actors. An actor may have behavior, which denotes the change of the fields of the actor over time. A transition that changes the story from one scene to the next one is triggered by the user interaction. APRIL provides built-in high-level user interaction commands, such as displaying a button in a particular stage of user interaction that can trigger a transition when the button is clicked. APRIL allows also to describe detection of the intersection of a pointer with the geometry of an actor. The language provides additional so-called "pseudo-interactions", such as timeout responsible for

automatically triggering or disabling certain transitions. Listing 3.1 presents an example of the APRIL code.

Listing 3.1: A snippet of the APRIL code.

```
<april xmlns="http://www.studierstube.org/april"
  xmlns:ot="http://www.studierstube.org/opentracker">
  <setup>
    <host name="showcase" ip="10.0.0.77">
    <screen resolution="1280 1024"/>
    <display screen="1" screenSize="fullscreen" stereo="true"
      worldSize="-0.4 0.3" worldPosition="0.098 0.162 0"
      worldOrientation="-0.1856 0.9649 0.1857 1.6057" mode="AR">
      <pointer mode="2D-RAY"/>
    </display>
       ...
  </setup>
  <cast>
    <actor id="ball" src="ball.apc">
      <input id="src" value="ball.iv"/>
      <input id="orientation" value="0 1 0 -1.57"/>
    </actor>
    ...
  </cast>
  <story>
    <sene name="empty" initial="true"/>
    <scene name="play"/>
    <transition event="enter" source="empty" target="play"/>
    ...
  </story>
  <behaviour scene="play">
    <entry>
      <set actor="ball" input="visible" to="TRUE"/>
      ...
    </entry>
  ...
  </behaviour>
</april>
```

Another noteworthy declarative language that was developed to model augmented reality environments was presented by Wojciechowski [160]. MRSML (Mixed Reality Scenario Modeling Language) is an XML-based language that allows content designers to specify both visual and behavioral characteristics of synthetic content. The MRSML language enables description of MR-Classes and MR-Objects. MR-Object is responsible for description of virtual objects, real objects, and mixed reality scenes. Similarly, MR-Classes embrace classes of virtual objects, real objects, and mixed reality scenes. MRSML takes advantage of basic elements of the object-oriented paradigm (OOP), such as: fields, operations, and inheritance. The MRSML language adopts elements corresponding to concepts of OOP, i.e., class, object, method, and attribute. These characteristics were extended with 3D geometry, images, video and audio clips, constraints on the attributes, interactive behavior, and aggregation of relationships of other MR-classes. Listing 3.2 shows a template for a class declaration in MRSML.

MR-Class defines a group of MR-Objects that have similar geometry, properties, media objects, behavior, relationships to other MR-Objects, and semantics. Various MR-Object instances of particular MR-Class may be initialized with different values of attributes. Thus, a presentation of different

MR-Objects may differ in visual, auditory, and behavioral aspects. The author used MR-Classes in education to model augmented reality learning scenarios.

Listing 3.2: The template of the MRSML class declaration.

```
<Class name="..." desc="..." super="..." virtual="..." category="...">
  <Attributes>
    <Design>
      ... // attributes for design interface
    </Design>
    <Creation>
      ... // attributes for creation interface
    </Creation>
    <Setup>
      ... // attributes for setup interface
    </Setup>
  </Attributes>
  <Constraints>
    <Constr cond="..."/>
      ... // constraints
    </Constraints>
  <Methods>
    <Method name="...">
      ...
    </Method>
    ... // methods
  </Methods>
  <Activities>
    <Activity name="...">
      ...
    </Activity>
    ... // activities
  </Activities>
  <Geometry>
    ... // geometry
  </Geometry>
</Class>
```

MacIntyre et al. presented an extension to spatial markup language for Google Earth and Maps (KML) [87]. The proposed KARML language supports the functionality required in mobile AR systems with the use of standard web technologies such as HTML5, CSS, and JavaScript. KARML allows to bind the presentation content and locations with the physical world. Standard KML already supports attaching COLLADA 3D presentation content to HTML code, however, the KARML extension overcomes some limitations. First of all, KARML adds a new element – `Balloon` to add control for the location, orientation, and scaling of balloon content. Furthermore, the language uses `orientationMode` and `scaleMode` elements letting the user to toggle billboarding and relative scaling modes respectively. Moreover, KARML adds a `locationMode` element, which enumerates "fixed" and "relative" modes. This element overcomes the KML limitation that latitudes and longitudes are only absolute references to degree coordinates. Using `locationMode` it possible to declare that some synthetic content is placed at the height of, e.g., 6 centimeters or meters depending on the units. Finally, KARML specifies the `Tracker` element allowing the user to use other sources of position information than sensors, including marker-based and markerless tracking. The KARML language is used by Argon AR Web Browser and

Listing 3.3: A snippet of the KARML code.

```
<Placemark>
   <name>MyPlacemark</name>
   <description>
      <![CDATA[<img style="width:300;height:200"
         src="http://myexample.com/content.png"/>]]>
   </description>
      <karml:Balloon>
      <locationMode units”=”meters targetHRef”=#”user>relative</locationMode>
      <location>
         <latitude>2.0</latitude>
         <longitude>0.0</longitude>
         <altitude>0.0</altitude>
      </location>
   <orientationMode>billboard</orientationMode>
   </karml:Balloon>
   ...
   <karml:Tracker>
      <karml:TrackerDevice>stbTracker</karml:TrackerDevice>
      <karml:TrackerDescription>
         <stbTracker:type>simpleId</stbTracker:type>
         <stbTracker:id>11</stbTracker:id>
      </karml:TrackerDescription>
   </karml:Tracker>
</Placemark>
```

was applied to numerous location-based projects [87, 132]. Listing 3.3 depicts a snippet of an example mark place described with the KARML language.

KARML is not the only attempt to extend KML for augmented reality applications. The Augmented Reality Markup Language (ARML) also extends KML with AR-specific structures. ARML is a descriptive, XML-based language focused on mapping geo-referenced Points of Interests (POIs) and their metadata. The language also permits to describe POI content providers. It adds markup extensions to support Wikitude browser features, such as `ar:provider` and `wikitude:info`. The `ar:provider` element is responsible for identifying a unique content provider, while `wikitude:info` describes additional information including thumbnail, phone, URL, email, and address [80]. An example of the KML placemark extended with additional information is presented in Listing 3.4.

Listing 3.4: A snippet of the ARML 1.0 code.

```
<kml xmlns="http://www.opengis.net/kml/2.2"
    xmlns:ar="http://www.openarml.org/arml/1.0"
    xmlns:wikitude="http://www.openarml.org/wikitude/1.0">

   <Document>
      <ar:provider id="mountain-example.com">
         <ar:name>Mountain Tours</ar:name>
         <ar:description>preferred mountain tours in the alps.</ar:description>
         <wikitude:providerUrl>http://mountain-example.com</wikitude:providerUrl>
         <wikitude:logo>http://mountain-example.com/logo.png </wikitude:logo>
      </ar:provider>
      <Placemark id="m">
         <ar:provider>mountain-example.com</ar:provider>
         <name>Gaisberg</name>
         <description>Description of Gaisberg</description>
```

```
16          <wikitude:info>
17             <wikitude:thumbnail>
18                   http://mountain-example.com/l.png
19             </wikitude:thumbnail>
20             <wikitude:phone>555-99432</wikitude:phone>
21             <wikitude:url>http://en.wikipedia.org/wiki/Gaisberg</wikitude:url>
22             <wikitude:email>info@mountain-example.com</wikitude:email>
23             <wikitude:address>...</wikitude:address>
24             </wikitude:info>
25          <Point>
26             <coordinates>13.11,47.81,1100</coordinates>
27          </Point>
28       </Placemark>
29    </Document>
30 </kml>
```

The next language – Augmented Reality Experience Language (AREL) – is a JavaScript binding of the *metaio* SDK's API in combination with static XML descriptions. AREL allows to script interactive AR applications based on common web technologies – HTML5, XML, and JavaScript [2]. Typically, an AR program based on AREL consists of two parts: a static XML description and a dynamic JavaScript code. The static part consists of a description of an AR scene and virtual objects. A `SceneOptions` element describes initial parameters of the AR scene, such as camera parameters, the environmental map of images, the material shader used for rendering, and information about how much annotations for location-based POIs should be moved up in pixels. A single `object` element contains the following data: a title, a thumbnail, an icon, a geo-location for this object, assets describing a 3D model, and parameters specifying optional visibility restrictions. AREL also allows to include additional parameters that should be passed to the dynamic part – the JavaScript code. Within JavaScript part, a programmer can implement interactions and behaviors of virtual objects, e.g., scaling the object.

Listing 3.5: A snippet of AREL code.

```
1  <results>
2     <arel>simpleAREL.html</arel>
3     <object id"="Tiger>
4        <assets3d>
5          <model>Assets/tiger.md2</model>
6          <texture>Assets/tiger.png</texture>
7          <transform>
8           <translation>
9              <x> 10.0</x><y> 0.0</y><z> 0.0</z>
10          </translation>
11          <rotation type="eulerdeg">
12             <x> 90.0</x><y> 0.0</y><z> 0.0</z>
13          </rotation>
14         </transform>
15         <properties>
16          <coordinatesystemid> 1</coordinatesystemid>
17         </properties>
18       </assets3d>
19    </object>
20 </results>
```

Last but not least, a redesigned version of the ARML – ARML 2.0 – language was proposed by the Open Geospatial Consortium (OGC) [86, 103]. ARML 2.0 does not extend ARML 1.0. The goal

of ARML 2.0 is to provide an interchange format in which a designer can describe virtual objects in AR scenes with their appearances and their so-called anchors related to the real world. Furthermore, ARML 2.0 allows to bind ECMAScripts [66] for dynamic modification of an AR scene, as well as interaction with a user. The ECMAScript bindings are described in the JavaScript Object Notation (JSON) format [34] allowing to specify event handling and animations. Moreover, the language uses concepts from Geography Markup Language (GML) [101] to describe geographical features. The current version of ARML 2.0 does not specify non-visual objects, such as sound and haptic feedback.

The language uses three top-level elements: `Feature`, `Anchor`, and `VisualAsset`. A `Feature` represents a real world object that should be overlaid with some computer-generated content. This element consists of metadata of the real world object, as well as at least one `Anchor`. An `Anchor` element defines a link between the virtual and the physical object. It can be a spatial location tracked by motion sensors of the device, or a visual pattern, such as a marker, a QR code, or an image.

ARML specifies two different types of Anchors. The first is an abstract class (`ARAnchor`), which can represent three classes of objects. The first one is `Geometry` representing a point, a line, or a polygon described with spatial coordinate tuples. The second one is a `Trackable` object – a visual pattern that is detected and tracked in 3D space. The third subclass of `ARAnchor` is the `RelativeTo` class describing a relation to other Anchor instances. The second type of Anchor is `ScreenAnchor` which describes a fixed location on the screen.

The `VisualAsset` element describes how a particular Anchor should be represented in an AR scene. The representation can be described as a 2-dimensional (such as a label specified through HTML elements, a colored area, a plain text, an image) or 3-dimensional object.

Listing 3.6: A snippet of AREL code.

```
<arml>
  <ARElements>
    <Image id="placemarkMarker">
      <ScalingMode type="custom">
        <minScalingDistance>10</minScalingDistance>
        <maxScalingDistance>1000</maxScalingDistance>
        <scalingFactor>0.4</scalingFactor>
      </ScalingMode>
      <width>20</width>
      <href xlink:href="http://www.myserver.com/myImage.jpg" />
    </Image>

    <ScreenAnchor id="infoWindow">
      <style>left: 0; width: 100%; bottom: 0; height: 25%</style>
      <assets>
        <Label>
          <conditions>
            <SelectedCondition>
              <listener>feature</listener>
              <selected>true</selected>
            </SelectedCondition>
          </conditions>
          <src><b>$[name]</b><br />$[description]</src>
        </Label>
      </assets>
    </ScreenAnchor>

```

```
28      <Feature id="goldenGateBridge">
29        <name>Golden Gate Bridge</name>
30        <description>Description...</description>
31        <anchors>
32          <anchorRef xlink:href="#infoWindow" />
33            <Geometry>
34              <assets><assetRef xlink:href="#placemarkMarker" /></assets>
35              <gml:Point gml:id="myPoint">
36                <gml:pos>37.818599 -122.478511</gml:pos>
37              </gml:Point>
38            </Geometry>
39        </anchors>
40      </Feature>
41    </ARElements>
42  </arml>
```

3.3. Semantic web technologies in AR

This section presents the role of semantic web and how it was applied to AR. The section introduces semantic web standards and popular serialization formats with examples. After that, applications of the semantic web technology in augmented reality systems is discussed.

3.3.1. Overview of semantic web technologies

Semantic web – a term proposed by Tim Berners-Lee – provides a universal framework that allows data to be shared and reused across application, enterprise, and community boundaries [24]. According to the World Wide Web Consortium (W3C) [146], the semantic web is a web of structured data, decoupling applications from data through the use of a simple, abstract model for knowledge representation. This abstract model releases the bilateral constraints on applications and data, letting both to evolve independently. As a consequence of using such model, any application that understands the model can consume any data source using this model [166].

Nowadays, the vast majority of information available on internet is unstructured, encoded in diverse text format such as HTML, plain text, PDF, RTF, doc, and docx. This is suitable for humans who can read, understand the important information, and use it to guide further knowledge discovery. However, the heterogeneity of information is indecipherable to machines and makes automated processing difficult. If the diversity of information can be encoded by content providers into structured collections of information and sets of inference rules that can be used to conduct automated reasoning, any application could access and use the rich array of data. Semantic web is intended to make machine-processable knowledge on the basis of provided data sets [24].

Resource Description Framework

A number of standards have been developed to enable semantic representation of web resources. The World Wide Web Consortium (W3C) devised the Resource Description Framework (RDF) [147], which is a primary standard for representing information about resources on the web. RDF provides a simple way to make statements about web resources. Each statement asserts a fact about a resource. A statement consists of three elements:

- a *subject* is a resource described by the statement,

- a *predicate* is a property of the subject,
- an *object* is a resource or literal value of the property.

A statement – which is the fundamental building block of semantic representation – is called a *triple*, because of the three parts. The subject in a triple corresponds to some "thing" – an entity for which a conceptual class is created. The predicate is a property of the entity. The object can be presented in two different ways: as an entity that can be a subject in other triples, and as the literal value such as a Boolean, number, string, and other types of values about a subject. In RDF, a literal value can have a language – for instance English, Polish, or Japanese. RDF conceptualizes anything and everything in the universe as a resource to avoid type ambiguities. A resource is simply anything that can be identified with unique International Resource Identifier (IRI) [141] including Universal Resource Identifiers (URI) [142].

In RDF, it is common to shorten IRIs by assigning a namespace to the base of IRI and using only characteristic part of the identifier. For example, it is widespread practice to use `rdf` as a moniker for the base IRI `http://www.w3.org/1999/02/22-rdf-syntax-ns#`, allowing predicates such as `http://www.w3.org/1999/02/22-rdf-syntax-ns#type` to be abbreviated as `rdf:type`. Listing 3.7 depicts two exemplary statements about Barack Obama encoded in RDF/XML format.

Listing 3.7: A snippet of RDF code.

```
<?xml version="1.0" encoding="utf-8" ?>
<rdf:RDF
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
      xmlns:dbo="http://dbpedia.org/ontology/"
      xmlns:dbp="http://dbpedia.org/property/" ...>

  <rdf:Description rdf:about="http://dbpedia.org/resource/United_States">
    <dbo:leader rdf:resource="http://dbpedia.org/resource/Barack_Obama" />
  </rdf:Description>

  <rdf:Description rdf:about="http://dbpedia.org/resource/Barack_Obama">
    <dbo:birthPlace rdf:resource="http://dbpedia.org/resource/United_States" />
    ...
  </rdf:Description>
```

The code can be read as the following sentences:

1. *United States* has the *leader Barack Obama*.
2. *Barack Obama*'s *birthplace* is the *United States*.

In the first statement, *United States* is the subject (http://dbpedia.org/resource/United_States). The word *leader* is the predicate. *Barack Obama* is the object (http://dbpedia.org/resource/Barack_Obama). In the second statement, the subject is *Barack Obama*, the predicate is *birthPlace*, and the object is *United States*. Figure 3.8 depicts visualization of the RDF code presented in Listing 3.7.

An RDF model is represented as a set of statements (triples) in which the same subjects and objects can be used in different triples. The RDF model forms a directed graph. Using the semantic web's terminology, subjects and objects correspond to vertices connected with directed edges (predicates). One of powerful properties of using graphs to model information is that having two or more separate graphs with a consistent system of identifiers for subjects and objects, it is relatively easy to merge the graphs. If a triple is written in both independent graphs, the two triples can be integrated together transparently. Figure 3.8 presents a visualization of exemplary RDF model composed of independent graphs of data.

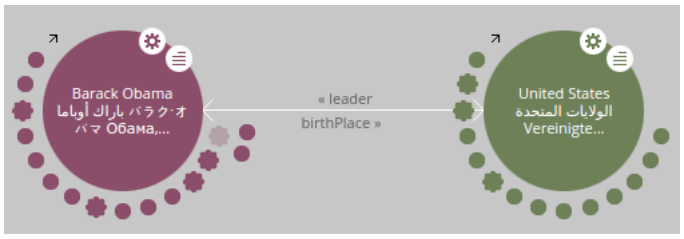


Figure 3.8: Visualization of two example RDF statements.

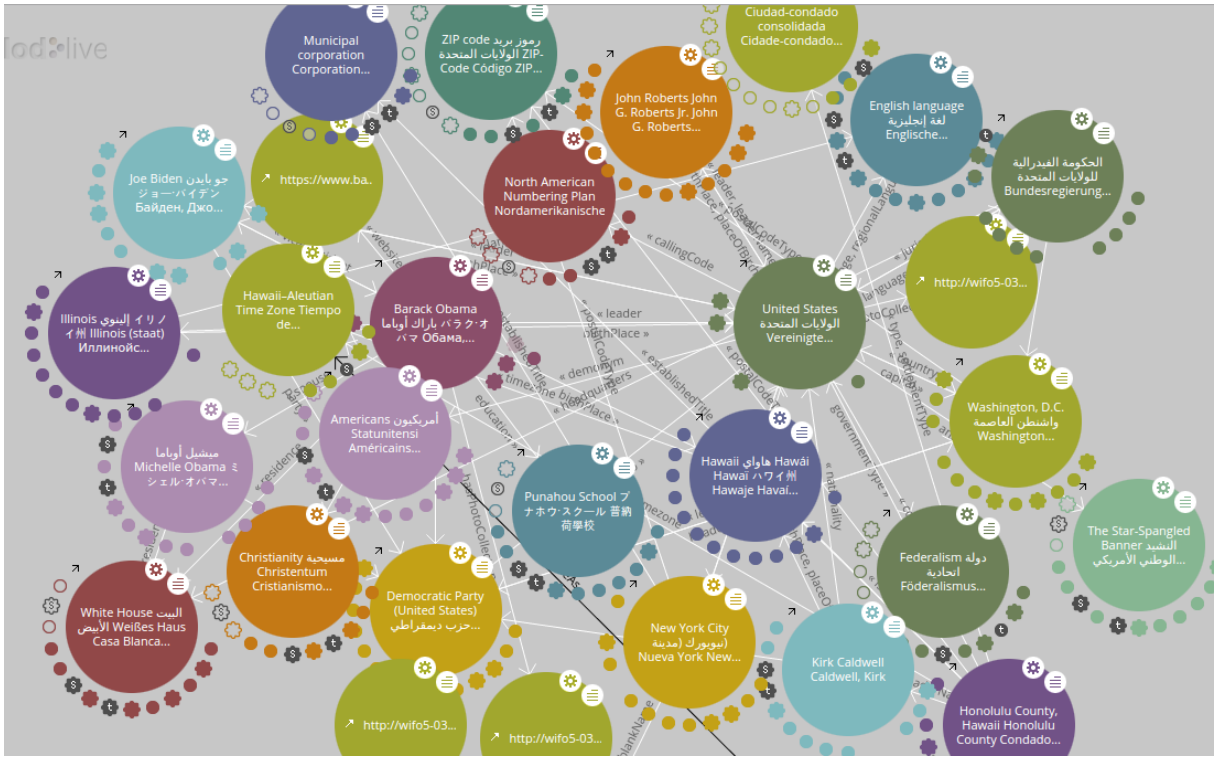


Figure 3.9: Visualization of independent graphs merged into one graph.

RDF serialization formats

Within this subsection five RDF serialization formats: N-Triples, Notation3 (N3), RDF/XML, RDF/JSON, and Turtle are shortly presented.

N-Triples

N-Triples [165] is a line-based, plain-text notation for encoding RDF graphs. Its original intent was for writing test-case data models while developing the RDF specification. However, it turned out that this format has proven to be popular as an exchange format for RDF data.

Each line of the triple encoded in N-Triples is a sequence of RDF terms representing the subject, the predicate, and the object followed by a dot. The principal components of an RDF triple are separated by white spaces. Objects literals are double-quoted strings that use the backslash to escape tabs, newlines, double-quotes, and the backslash character. Optionally, string literals can specify their language when followed by `@lang`. Also, the datatype IRI of the literal value has to be followed by `^^type`.

Listing 3.8 depicts an example of an RDF triple encoded in the N-triples notation.

Listing 3.8: An example of RDF triples written in N-triples format.

```
1 <http://dbpedia.org/resource/United_States> <http://dbpedia.org/ontology/leader>
```

```
<http://dbpedia.org/resource/Barack_Obama> .

<http://dbpedia.org/resource/Barack_Obama> <http://dbpedia.org/ontology/birth_place>
<http://dbpedia.org/resource/United_States> .
```

Notation3 (N3)

While the N-Triples format is conceptually easy, the main disadvantage is the repetition of redundant information, which as a consequence takes additional time to transmit and parse. While it is not a problem when working with small sets of data, the redundant data becomes problematic when working with large amounts of data. Notation3 (N3) [164] significantly reduces number of characters used in N-Triple by providing short symbols representing repeated nodes. N3 defines an IRI prefix (`@prefix`) – similarly to the XML namespace mechanism for generating short names for nodes, so-called qualified names (qnames) – and identify entity IRIs relative to a set of prefixes declared at the beginning of the document.

The statement presented in Listing 3.9 allows to shorten the absolute IRI for *United_States* from *http://dbpedia.org/resource/United_States* to `dbr:United_States`.

Listing 3.9: Shortening an absolute IRI with N3 format.

```
@prefix dbr: <http://dbpedia.org/resource/> .
dbr:United_States ...
```

Continuing the example with Barack Obama, two sample statements encoded in N3 are presented in Listing 3.10. The presented description significantly reduces the size of the document when comparing N-Triples example from Listing 3.8.

Listing 3.10: Using @prefix to shorten repetitive IRIs in the RDF triple.

```
@prefix dbr: <http://dbpedia.org/resource/> .
@prefix dbo: <http://dbpedia.org/ontology/> .

dbr:United_States dbo:leader dbr:Barack_Obama
dbr:Barack_Obama dbo:birth_place dbr:United_States
...
```

RDF/XML

Another format used for representation of the RDF model is RDF/XML [148]. The RDF/XML notation allows to encode RDF graphs in XML terms – element names, attribute names, element contents, and attributes values.

The top-level element of the RDF graph written in RDF/XML is used to define XML namespaces used throughout the document. The RDF IRI reference is determined by appending the local name part of the XML qualified name after the namespace name (IRI reference). This approach allows to shorten the RDF IRI references of all predicates and some nodes.

An RDF graph can be considered as a collection of paths of the form node - predicate - node - predicate - node - predicate, etc. In RDF/XML these turn into sequences of elements inside elements which alternate between elements for nodes and predicates. This is called a series of node/predicate stripes.

The `rdf:Description` element always starts a path. The IRI reference of the subject is specified as a value of the `rdf:about` attribute. Each predicate is specified as a child element of

the `rdf:Description` node. The `rdf:resource` attribute value contains the IRI of the object. Listing 3.7 depicts two example statements about Barack Obama encoded in the RDF/XML format.

RDF/JSON

The RDF/JSON format allows an RDF graph to be written in a form compatible with the JavaScript Object Notation (JSON) [34]. RDF/JSON serializes RDF triples as a series of nested data structures. An RDF/JSON document consists of a single JSON object called the *root* object. Each unique subject in the set of triples is represented as a key in the root object. No key may appear more than once in the root object. Each root object key's value is a JSON object whose keys are the URIs of the predicates occurring in triples with the given subject – known as predicate keys. No predicate key may appear more than once within a single object. An array of JSON objects, representing the object of each serialized triple, is the value of each predicate key.

In general, a triple written in the RDF/JSON format (subject `S`, predicate `P`, object `O`) is serialized as an instance of the template structure presented in Listing 3.11.

Listing 3.11: A template for the RDF/JSON notation.

```
1 { "S" : { "P" : [ O ] } }
```

The object of the triple `O` is described as a further JSON object with four keys: `type`, `value`, `lang`, and `datatype`. The value of `type` key can be written as 'uri', 'literal' or 'bnode'. Next, the value of the `value` key is the URI of the object. The value of the `lang` is the language of a literal value. Finally, the value of the `datatype` is the datatype URI of the literal value. The 'lang' and 'datatype' keys should only be used if the value of the 'type' key is "literal". All keywords defined in RDF/JSON format are case sensitive, and must be written lowercase. Listing 3.12 presents an example of an RDF triple encoded in the RDF/JSON format.

Listing 3.12: An example of the RDF triple written in the RDF/JSON format.

```
1 {
2       "http://dbpedia.org/resource/United_States" : {
3             "http://dbpedia.org/ontology/leader" : [ {
4                   "value" : "http://dbpedia.org/resource/Barack_Obama",
5                   "type" : "uri" } ]
6   }
7 }
```

JSON-LD

JSON-LD is an alternative to the JSON-LD format described in the previous subsection [163]. JSON-LD is intended to use RDF data in web-based programming environments, to create interoperable web services, and to persist Linked Data in JSON-based storage mechanisms. This format is recommended by W3C.

JSON-LD serializes RDF data as JSON objects via the use of IRIs (as opposed to URIs used in RDF/JSON). It provides a universal mechanism to disambiguate keys shared among different JSON documents by mapping them to IRIs via a context. The format allows to refer to a JSON object located on a different site on the web as a value in another JSON object. A single JSON-LD document can be used to express one or more RDF graphs, such as social networks, in a single document. It is also possible to annotate values of properties with their native language meanings (Listing 3.13).

Listing 3.13: An example of the language map expressing a property in two languages encoded in JSON-LD.

```
{
...
        "occupation":
        {
              "pl": "Prezydent",
              "en": "President"
        }
...
}
```

RDF Schema

RDF Schema (RDFS) provides a data-modeling vocabulary for describing classes and properties of RDF resources [153]. RDFS is a semantic extension of RDF providing mechanisms for describing groups of related triples and relationships between resources. RDF Schema is written in RDF and is built on the RDF resources that can be used to describe other RDF resources in application-specific domains.

Although useful in many applications, RDFS lacks a number of important features, such as: local scope properties (it is not possible to declare range restrictions that apply to some classes only; possibility to declare disjointness of classes; Boolean combinations of classes (union, intersection, and complement); cardinality restrictions; special characteristics of properties (transitive, unique, and inverse). These limitations are solved in the OWL language described below.

The Web Ontology Language 2 (OWL 2)

The Web Ontology Language 2, informally OWL 2 [150], is an RDF language used for the formalization of the meaning in semantic web. The language is an extension and revision of previous specification [154] and RDFS [153], and it enables more powerful reasoning and inference mechanisms over relationships between classes and individuals. OWL 2 has a richer vocabulary description language for describing classes and properties than RDFS. The language uses the RDF/XML format and is the current W3C standard for defining semantic web schemes.

OWL 2 enables developing a contract for meaning – an ontology that is a formalized vocabulary of terms, often covering specific business logic and shared by a community of users. An ontology enables to specify which entities will be represented, how they can be grouped, and what relationships exist between them. The knowledge expressed in OWL can be reasoned with by computer applications either to verify the consistency of that knowledge or to make implicit knowledge explicit. The more precise the ontology, the greater the potential of understanding of how the data can be used. However, if the ontology is overly complex, then it can become confusing, intricate, and problematic to use, maintain, and extend by humans.

In OWL 2, objects are denoted as individuals, categories as classes, and relations as properties. Properties in OWL 2 are subdivided into Object properties, Datatype properties, and Annotation properties. Object properties relate objects to objects (like markers to virtual objects), while datatype properties assign data values to objects (like a camera resolution to a camera). Annotation properties are used to describe information about (parts of) the ontology itself (like the author and creation date).

In order to model the information that two individuals are interconnected by a certain property, *domain* and *range* properties are used. These two properties allow to draw conclusions about the individuals themselves. For instance, the statement "Epson Moverio BT-200 has cameraXYZ" implies that the 'Epson Moverio BT-200' individual (domain) is related via "hasCamera" property the 'cameraXYZ' individual (range). The statement that two individuals are related via a certain property

carries implicit additional information about these individuals. The above-presented example also implies that 'Epson Moverio BT-200' is a type of the *Wearable Glasses* class and 'cameraXYZ' is a type of the *Camera* class.

Figure 3.10 depicts an example of the domain and range of the ex:hasCamera property, which indicates classes of objects that have a camera.

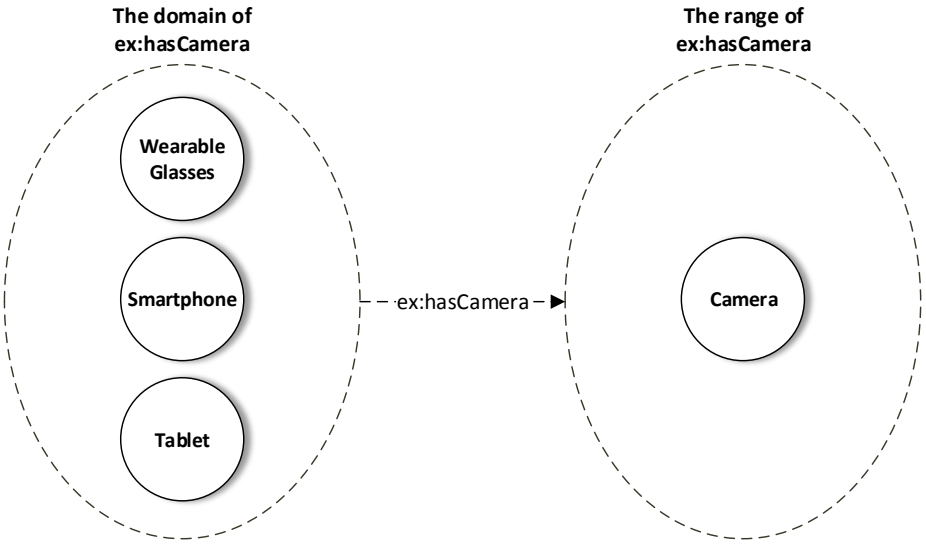


Figure 3.10: The `domain` and `range` of the property expressing *hasCamera*.

SPARQL

SPARQL – *SPARQL Protocol and RDF Query Language* – provides a standardized query language for RDF graphs (similarly to SQL which defines a query language across relational database systems) [152]. SPARQL gives the ability to formulate queries ranging from simple graph pattern matching to complex queries. There are other RDF query languages, such as RDQL (RDF Data Query Language) [149] and SeRQL (Sesame RDF Query Language, pronounced "circle") [131]. Due to the W3C standardization and the wide community support, SPARQL can be considered as the main query language that helps to process RDF graphs.

It is noteworthy that SPARQL is both a query language and a protocol. The protocol is used to specify how SPARQL client (e.g., accessible via a web browser) communicates with so called endpoint (processor) both in an abstract sense and using a concrete implementation, e.g., based on WSDL.

SPARQL provides four forms of queries `SELECT`, `CONSTRUCT`, `ASK`, and `DESCRIBE`. These clauses attempt to find solutions to a graph pattern, and all share similar constructs. A `SELECT` query identifies a subset of the variables used in the graph pattern whose bindings are returned in a table format. It is comparable to SQL's `SELECT` statement. The `SELECT` keyword binds RDF terms, such as blank nodes, IRIs, or literals) to variables based on the given graph pattern. Bindings are returned as result sets in a form of an array. Next, a `CONSTRUCT` query extracts information as a valid RDF graph. Next clause – `ASK` – allows to test whether a pattern can be found in a graph. It corresponds to the SQL `WHERE` keyword and returns a simple Boolean as a result indicating whether the graph provides a solution for a query. Each of these are followed by a `WHERE` clause that specifies the graph pattern to match as a collection of triples. Variables in the triple pattern start with a question mark (?) or a dollar sign ($).

The version 1.1 of the SPARQL specification enables the following operations on RDF graphs: insert (`INSERT`), update (`DELETE INSERT`), and remove (`DELETE`) triples from RDF graphs.

Listing 3.14 presents a snippet of the SPARQL query for finding all landlocked countries with a population less than ten thousand and written in English. Table 3.2 depicts the result of the query run on a DBpedia SPARQL endpoint [19, 37].

Listing 3.14: A snippet of simple SPARQL query processed by a DBpedia service.

```
PREFIX type: <http://dbpedia.org/class/yago/>
PREFIX type: <http://dbpedia.org/class/yago/>
PREFIX prop: <http://dbpedia.org/property/>
SELECT ?country_name ?population
WHERE {
   ?country a type:LandlockedCountries ;
          rdfs:label ?country_name ;
          prop:populationEstimate ?population .
   FILTER (?population < 100000 && langMatches(lang(?country_name), "EN")) .
} ORDER BY DESC(?population)
```

Table 3.2: The result of the SPARQL query presented in Listing 3.14

country_name	population
"Andorra"@en	85458
"Liechtenstein"@en	37340
"San Marino"@en	32576
"Vatican City"@en	842

3.3.2. Applications of semantic web in augmented reality

Schmalstieg and Reitmayr argued that ubiquitous augmented reality systems require independence of the data model from specific applications, and that to deal with it a semantic model of geo-referenced data can be used [127]. The authors derived a data model which allows a suitable degree of semantic reasoning for mobile AR and described how it can be used in urban navigation. For instance, the authors explained general classes, such as *IsInteresing*, *SubstituteFor*, and *neighboursWith*, which can be used for the presentation of objects that are "semantically" related to the target objects, even if targets are not directly visible in a user interface.

In [114], Reynolds et al. discussed future directions for mobile augmented reality applications – in particular – how linked data [26] can be used in mobile AR browsers for enhancing the reality with information about local points of interest, such as historical sights, nearby bus stops, and cafés. The authors argue that techniques of the semantic web can be used for dynamic selection and integration of data from different sources. Furthermore, the use of a cloud of linked open data, such as GeoNames, LinkedGeoData or DBpedia, can provide a wide range of contextual information for mobile AR browsers. The authors also state that the browsing experience with linked data is similar to that we know when we surf the Internet using standard web browsers.

Hervás et al. presented [64] an augmented reality information system for elderly or dependent users by describing context information with the semantic web and QR codes. The authors developed a model called Mobile Augmented Reality Model (MARM) that represents the information to be superimposed onto a view of physical objects surrounding a user. MARM requires data provided by an accelerometer

and a digital compass to adapt information presentation in the user interface. QR codes are used to provide RDF data corresponding to the user's location. The model and the implemented application is suitable for users that need to be guided in their daily activities as it occurs in typical Ambient Assisted Living (AAL) scenarios, e.g., how to prepare a meal, which medicament the user has to take or when is the next appointment with the doctor (Fig. 3.11).

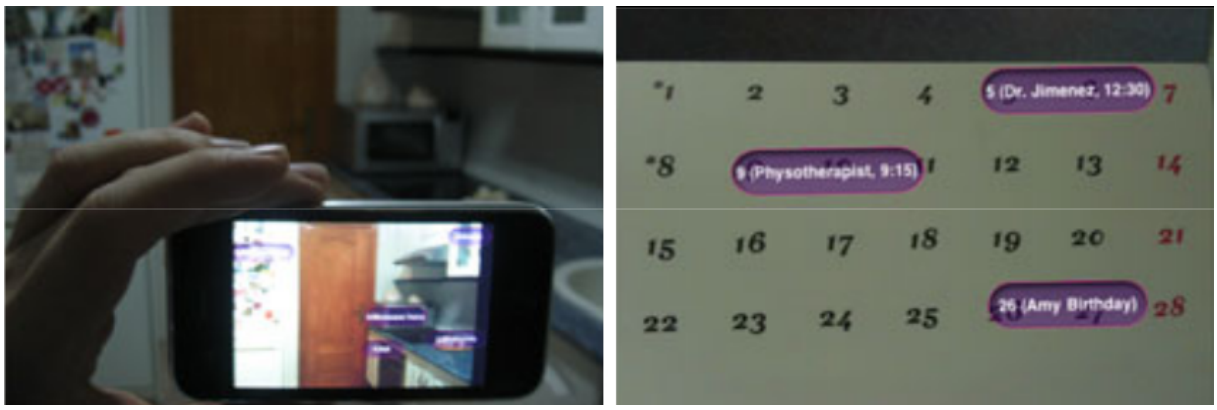


Figure 3.11: Hervás et al. solution in AAL scenarios use [64].

A new approach for indoor navigation using semantic web augmented reality technologies was presented by Matuszka et al. [89]. Similarly to the solution presented by Hervás et al., QR codes were used for recognizing coordinates of indoor locations that are further processed by the server responsible for providing semantically described data (passages, corridors, exit, etc.) associated with the QR code. Additionally, the server computes the possible paths between two different locations using SPARQL queries. Moreover, QR codes are also used for visualization of 3D arrows indicating the proper direction to the chosen location.

Another Matuszka's work [91] presents a location-based augmented reality application for exploring well-known Hungarian Kerepesi cemetery with the use of semantic information about deceased people. The application calculates distance between the GPS coordinates of graves and the user's position and navigates the user to the correct location. When the selected grave is in the field of view of the camera, the application runs SPARQL queries to retrieve information about the deceased person and shows a 2D graphical information overlay on the view of the grave. In [90] Matuszka and Kiss explain an architecture of the semantically enriched location-based augmented reality browser. On the basis of the user's geographical location, a client application retrieves RDF data from DBpedia database. The AR interface visualizes 2D annotations representing selected RDF data.

Nixon et al. [100] demonstrated the *SmartReality* platform that combines augmented reality and semantic web technologies in the entertainment domain. The goal of the presented AR system is to use linked data to provide information about clubs, artists, and music concerts that take place around the user's location. *SmartReality* mediates between the Android-based client, which is capable to overlay 2D annotations onto concert posters, and web-based data and services, providing the most appropriate content and services for display to the user, based on available metadata.

Applications of semantic web in augmented reality systems for cultural heritage have been presented in [12] and [157]. Van Aart et al. developed a location-based outdoor application that combines Linked Data with domain-specific cultural heritage content [12]. The mobile application explores and visualizes RDF data provided by a back-end server on the basis of the user's GPS location. The back-end server aggregates, harvests and merges sets of triples that describe the location of the user and sends them

Figure 3.12: Annotating additional information on a view of a grave in the application presented by Matuszka [91].

back to the client application. Although the authors presented the augmented reality user interface that visualizes data, it is not clear how the visualization is registered in the view of the real world objects.

Walczak et al. [157] developed an indoor AR system for enhancing museum paintings on the basis of domain-specific ontologies developed and maintained by domain experts. The system permits museum visitors to acquire additional information about paintings presented in exhibitions. The client application recognizes and tracks paintings annotated by museum curators and highlights specific areas of the paintings (e.g., painted people or objects) on which more information is available. Museum paintings are described with the semantic AR representations consisting of RDF statements and SWRL rules stated on objects, which are instances of semantic concepts specified in the ontologies.

Figure 3.13: Semantically described paintings of Polish monarchs presented on two different classes of devices [157].

The domain knowledge base is used to generate final AR scene descriptions that is used by a mobile application. Figure 3.13 depicts a painting containing portraits of three Polish monarchs, who are indicated by color circles – depending on their dynasties. Each circle represents an augmentation area, on which more information can be provided. By tapping any of the augmentation areas, a user can

retrieve the associated information, which is presented as a textual object above the painting. Figure 3.13 presents semantically described painting enhanced with AR information about Polish monarchs.

Zander et al. presented a mathematical model that exploits locally gathered sensor data in combination with geographical information provided by three Linked Data sources in order to visualize the topologicaly highest elevation or ridge atop a mountain in AR interface [167]. The authors conducted several experiments in order to evaluate the performance of the solution. The results obtained by the authors demonstrated that the implemented AR system is effective and the proposed mathematical model can be used in real-world cases to visualize LOD data in location-based AR interfaces.

Vert and Vasiu demonstrated an augmented reality web application for tourists that integrates Linked Open Data and Romanian government data [145]. The application renders 2D pinpoint icons overlaid on a view of tourist places located around a user and displays complete information about the selected POI. In that work, the authors also point out the challenges that have to be overcome when developing such applications, which fuse diverse data sources. Figure 3.14 depicts three screenshots taken from the Vert and Vasiu's application.

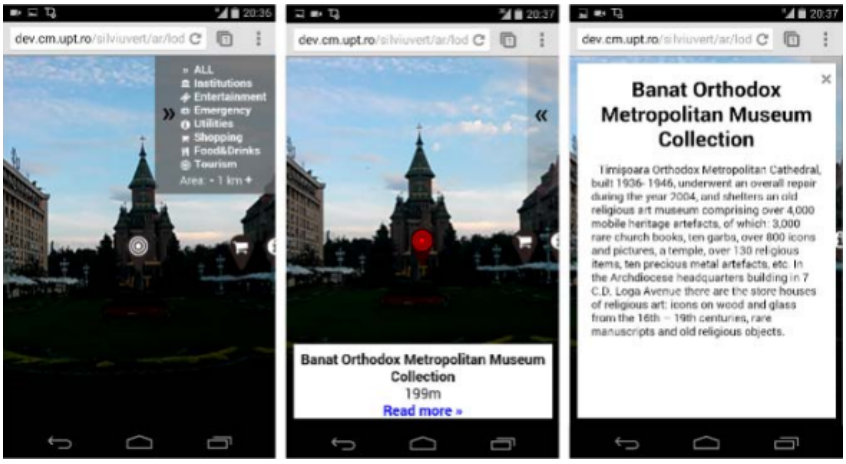


Figure 3.14: Augmenting POIs around the user with the LOD data [145].

Ruta et al. presented an augmented reality mobile framework for semantically-enhanced tourist places discovery [124]. The solution allows users to see a 2D overlay of markers representing points of interests on a view displayed in a mobile device. The authors implemented a location-based AR browser, which exploits semantically enhanced OpenStreetMap cartography and executes semantic queries on the basis of the user's preferences and relevant POIs in the user's surroundings. Figure 3.15 depicts the AR interface developed by Ruta et al.

Aydin et al. presented an architecture for location-based augmented reality applications enhanced with semantically described historical buildings [22]. The authors developed a data model that takes into account thematic, temporal, and architectural information about historical buildings. With this approach, a user is able to get additional information about religious, residential, and commercial buildings and see in an AR interface how geometry of these places have evolved over time. Although presented solution seems very promising to facilitate the user's surroundings discovery, it is not clear how 3D models of historical buildings can be properly aligned with the view of real buildings using only location-based tracking techniques in AR applications.

Figure 3.15: Semantically enhanced AR browser for tourists presented in [124].

3.4. Limitations of existing methods of modeling AR environments

Although existing AR tools provide relatively quick and convenient methods for developing AR applications, they have significant limitations. Existing AR platforms support mainly two forms of augmentation: *directional augmentation* – based on relative geographical position and orientation of the user's device and fixed coordinates of specific points of interest, and *image-based augmentation* – based on image matching and tracking. The advantage of image-based augmentation results from the fact that synthetic content is directly aligned with a view of real-world objects. However, due to limitations of the available image matching algorithms, image-based AR applications are built independently for specific purposes. The current AR applications are developed independently based on different approaches, different data models, and specific algorithms. Most studies in the AR field have focused on developing software responsible for image recognition and implementation of applications offering specific content and functionality – in most cases, run in a specific hardware configuration. Vast majority of currently available applications and tools do not allow to share AR content between users and do not enable combining AR content coming from different sources. Neither it is possible to build borderless ubiquitous AR applications, which allow a user to cross boundaries of service availability.

Currently, software packages do not provide support for a rich user's context, such as outside or inside location, preferences, time and date, and device capabilities. Despite the increasing prevalence of AR tools, there is still a lack of methods that allow to search, combine, and present rich multimedia content coming from distributed sources, including the user's context. For instance, it would be useful and beneficial to use the same application to discover interesting places in the town by using services offered by the city, while after entering a shopping centre – with the use of the same application – it would be also possible to obtain information such as prices, opinions, video movies and sounds, presenting goods offered by stores, which are of interest to a user. Currently, each of these functions is feasible, but requires a different, independent application.

In most existing AR applications, content, data, and interactions are tied with a specific set of multimedia objects and cannot be reused in a runtime with other multimedia objects retrieved from external sources. As a result, users have to install many different AR applications to experience different AR presentations. Per analogy, in the today's World-Wide Web it is unimaginable to have to install a specific web browser to be able to visit a specific website. Current web browsers are based on a common set of standards, which allows them to present different kinds of content from various sources. The situation is different in case of AR applications, which in most cases are not compatible at the software and content level.

In case of using easy-to-use AR development software, AR presentations become dependent on the proprietary mobile AR browser – in most cases – only providing the function of displaying virtual objects. Moreover, the user interface functionality is limited, not allowing sophisticated interaction with the presented content. Most of visual tools are appropriate for non-technical users without programming skills who can quickly prototype AR applications, e.g., to present an idea of an AR experience. But when there is a need to model some specific interactions between a user and virtual objects, a designer is limited to only some basic functions of the software. Thus, to build an AR application with a specific interface allowing refined interactions and to use the capabilities of modern mobile devices, it is necessary to use low-level AR software libraries/frameworks.

The overview of augmented reality systems supported by semantic web technology presented in previous section shows that researches in most cases concentrated on creation of AR interfaces that are capable to overlay 2D annotations enriched with data coming from specific data servers, such as DBpedia. Retrieved LOD data, based on the user's context (taking into account a GPS location or associated with a QR code) are presented using static 2D textual components. These systems do not permit users to experience rich contextual augmented reality presentations based on a 2D and 3D multimedia content. Moreover, presented systems use specific semantic web ontologies to specific applications. Thus, to create another AR application enhanced with semantic web techniques, there is a need to develop yet another specific semantic data model. Last, but not least significant drawback of the presented works is lack of any quantitative evaluation of developed solutions in order to prove their applicability to large-scale contextual distributed environments.

3.5. Summary

In this chapter, selected augmented reality development tools have been reviewed. The presented solutions, by means of which it is possible to build AR applications, have been divided into two groups. The first group consists of low-level software libraries and frameworks providing the highest level of flexibility, but requiring advanced technical skills from developers. The second group comprises high-level integrated AR design tools and AR programming languages enabling non-technical users to create AR content in an easy way, but the created content in most cases is simplistic. Although developers and content designers can use a variety of existing tools to implement AR experiences, created AR content is closely bound up with the platform-depended application – not allowing its use in other situations where the user's context dynamically changes.

This chapter also explored the role of semantic web technology and its application within AR systems. To create an open environment of AR resources that can be combined into contextual AR presentations, there is a need of changing strategy of implementing AR systems. This dissertation addresses the problem of describing, designing, searching, interpreting, combining, and presenting independent resources that jointly form interactive AR presentations, taking into account context – the location (both indoor and outdoor), time, type of device and its capabilities, and users' preferences. This dissertation tries to answer the still open question how to enable efficient modeling of large-scale contextual distributed AR environments that will overcome the discussed drawbacks.

4. CARE – Contextual Augmented Reality Environment

Within this chapter, the *Contextual Augmented Reality Environment* (CARE) approach is presented. First, the general concept of dynamic composition of augmented reality experiences, based on resources offered by distributed independent providers of AR content and data, is described. Second, the formal model of the CARE environment is discussed. Third, the CARE architecture for distributed AR services is presented. Fourth, the *Semantic Augmented Reality Ontology*, which is used to model CARE environments, and its six sub-ontologies are described. Fifth, an algorithm of the *Semantic Discovery and Matching Method* that is responsible for discovering and matching contextual elements forming AR presentations in CARE is presented. Next, a new high-level programming language, called *Contextual Augmented Reality Language*, is presented which enables modeling contextual augmented reality environments. Finally, a summary of the chapter is provided.

4.1. The CARE approach

To deal with limitations discussed in Section 3.4, a new approach to building AR systems is required. The CARE approach – which stands for *Contextual Augmented Reality Environment* – is the main contribution of this dissertation. CARE goes beyond the current state of the art in modeling augmented reality experiences by providing uniform cross-application contextual access to AR content and services. The main novelty of CARE lies in avoiding fragmentation of augmented reality functionality between multiple independent applications and dynamic integration of various information services into unified, contextual and personalized AR interfaces. In CARE, augmented reality presentations are built based on the user's context, which may include such elements as outdoor and indoor location, date, time, device type and capabilities, as well as user's preferences. Figure 4.1 presents the general concept of the CARE approach, in which AR presentations are dynamically composed of elements available in the user's context and coming from independent content- and data providers.

A contextual approach to AR application development is a necessity to enable access to a variety of independent data sources, guarantee scalability and provide for seamless operation when the context changes. In CARE, the same AR application can be used to display elements coming from different content and data providers. Thus, a user does not need to install a new application to experience another AR presentation. Moreover, CARE handles updating of content and data in the run-time, without requiring changes of the AR browser application. Hence, there is no need to update the AR application's code in the case when the available augmentations or a user's context change. Similarly as in web browser applications, such as Chrome or Firefox, there is no need to switch between applications when a user goes from one website to another.

To permit building dynamic AR presentations, the CARE Architecture of distributed augmented reality (AR) services is proposed [123]. The architecture is based on the Service Oriented Architecture (SOA) paradigm, which enables building distributed systems that provide application functionality as

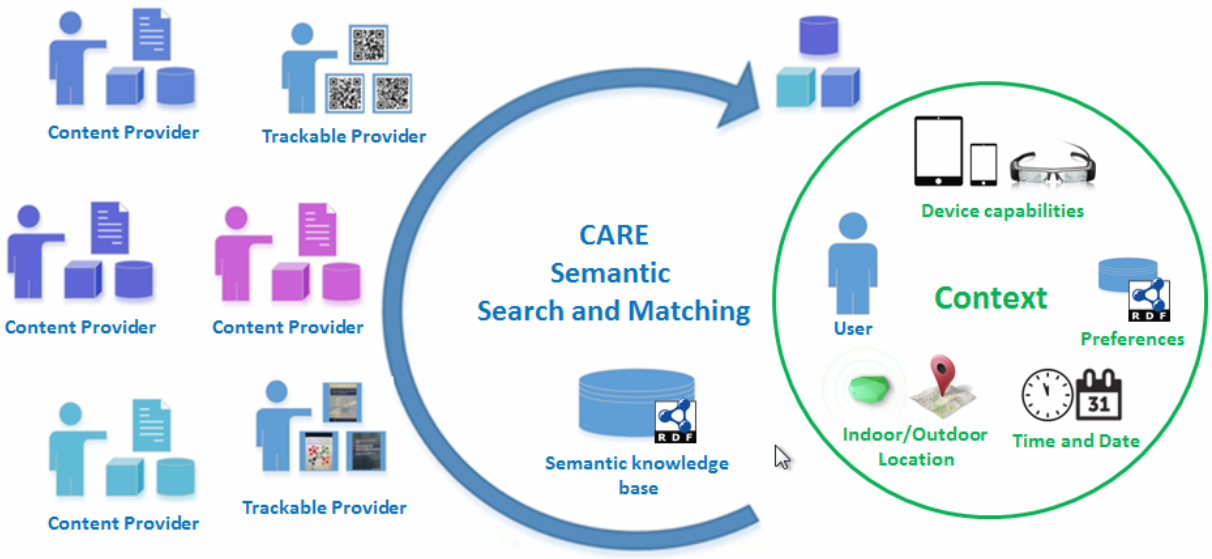


Figure 4.1: Dynamic composition of AR presentations in a CARE environment based on the user's context.

services to either end-user applications or other services. The CARE Architecture enables dividing responsibility between loosely coupled semantic AR services distributed on the internet.

The dynamism of composing AR presentations using data coming from multiple independent sources relies on the *Semantic Discovery and Matching Method* (SDMM). The proposed method is responsible for selecting content and data, which best meet given criteria, and which together form an AR experience. SDMM searches through semantic knowledge base designed with the use of the CARE model ontology, called *Semantic Augmented Reality Ontology* (SARO) [122]. SDMM and SARO are based on the semantic web standards (RDF, RDFS, and OWL 2), which permit creation of statements (facts and rules) that describe ubiquitous contextual augmented reality environments. These elements of the CARE approach are presented in detail in following chapters.

As an application protocol, a new high-level declarative language, called *Contextual Augmented Reality Language* (CARL), is proposed. The language enables modeling AR presentations whose elements come from diverse and distributed content- and data providers [115, 120, 121]. The CARL language – *Contextual Augmented Reality Language* – is designed to support dynamic composition of complex interactive AR presentations. In CARL, AR presentations consist of four independent semantically described elements:

- *Content objects* – a set of virtual content objects including visual and auditory data provided by the available information/content sources (e.g., 3D models, audio, and video);
- *Data objects* – a set of data objects indicating numerical or textual values;
- *Trackables* – a set of visual markers, indicating the elements (views) of reality that can be augmented;
- *Scenarios* – a description of actions and events occurring in particular AR presentation. It also characterizes various forms of possible interactions between a user and content objects in a particular AR presentation.

These four elements are independent and may be offered by different content- and data providers in a distributed architecture. For instance, trackable markers may come from municipal services, content and data objects from AR service providers, while scenarios from application developers. Only in such

heterogeneous environment, practical ubiquitous AR environments may be realized. A comprehensive description of the CARL language with examples is provided in Section 4.6.

To summarize, the CARE approach consists of the following elements:

1. *CARE Architecture of Semantic AR services*,
2. *Semantic Augmented Reality Ontology*,
3. *Semantic Discovery and Matching Method*,
4. *Contextual Augmented Reality Language*.

4.2. Formal model of CARE

This section presents a formal model of the Contextual Augmented Reality Environment and its elements associated with the CARE approach. A precise description is based on set theory and functions.

Definition 1 - Augmented Reality Resource Class

An Augmented Reality Resource Class (ARRC) – is an abstract class of content and data elements forming contextual augmented reality presentations.

Further, within this dissertation, instances of classes subclassing ARRC are referred to as AR resources. In general, AR resources come from diverse and distributed AR service providers. AR resources are identified by unique International Resource Identifiers (IRIs).

Definition 2 - Content Object Class

A Content Object Class (COC) – is an abstract class representing multimedia components that have their manifestations in the form of: 3D models, images, videos, and sounds.

COC is a subclass of ARRC. Instances of COC are referred to as content objects, or *co* in short.

Let $M = \{m_1, m_2, \ldots, m_i\}$ be a finite set of 3D models, $G = \{g_1, g_1, \ldots, g_j\}$ be a finite set of images, $V = \{v_1, v_2, \ldots, v_k\}$ be a finite set of videos sequences, and $S = \{s_1, s_2, \ldots, s_l\}$ be a finite set of sounds.

A content object is a 4-tuple consisting of subsets of the M, G, V, S sets:

$$co = (M, G, V, S), \tag{4.1}$$

where:

- M is the set of 3D models assigned to *co*,
- G is the set of images assigned to *co*,
- V is the set of videos assigned to *co*,
- S is the set of sounds assigned to *co*.

The set of content objects that can be arranged in a 3D space to augment real-world views is denoted as follows:

$$CO = \{co_1, co_2, ..., co_n\}, \tag{4.2}$$

where n is the total number of content objects in a CARE environment.

The content objects may be simple or complex – consisting of a single or multiple multimedia components m, g, v, s.

Content objects are generic, i.e., they do not depend on the client application. The AR browser application uses the available AR resources (including content objects) provided by various external AR service providers. Moreover, COs are independent of operating systems, end-user device's type, software, and hardware. COs are dynamically selected and composed into contextual AR presentations on the basis of the user's context. Content objects are loaded and rendered in the AR browser at runtime without a need of changing the source code of the application.

A content object can be reused in multiple AR presentations. For instance, three-dimensional interactive commercials added to each AR presentation in a particular context, can be reused multiple times.

Examples of content objects include cinema video trailers superimposed onto movie posters, graphical charts showing aggregated users' opinions, 3D interactive maps presenting the locations of events, pictures depicting faces of university lectures, or simply sounds directing user's attention.

A content object may be represented within the *Contextual Augmented Reality Language* (CARL) – using the <*ContentObject*> element.

Definition 3 - Data Object Class

A Data Object Class (DOC) – is an abstract class of objects representing numerical and textual data that can be used to compose contextual AR presentations.

Conceptually, DOC is a subclass of ARRC. Instances of DOC are referred to as data objects or *do*. A set of data objects that can augment real-world views is denoted as:

$$DO = \{do_1, do_2, ..., do_n\}. \tag{4.3}$$

Let $DO_N = \{n_1, n_2, \ldots, n_k\}$ be a set of numerical objects, $DO_T = \{t_1, t_1, \ldots, t_l\}$ be a set of textual objects. Formally, DO is a sum of DO_N and DO_T:

$$DO = DO_N \cup DO_T, \tag{4.4}$$

$$\forall do_i \in DO : do_i \in DO_N \vee do_i \in DO_T, \quad i \in < 1, n > . \tag{4.5}$$

The elements of DO may refer to numerical or textual information such as prices, discounts, duty hours, titles, names, surnames, date and time, etc. Data objects may change over time due to the fact that, e.g., prices can fluctuate, duty hours can vary during different times of year, titles can be modified, etc.

A data object may be represented in CARL using the <*DataObject*> element.

Definition 4 - Trackable Object Class

A Trackable Object Class (TOC) – is an abstract class of representations of elements of the real-world environment that can be recognized and tracked by sensor-based software.

Conceptually, TOC is a subclass of ARRC. Instances of TOC are referred to as trackables or trackable objects.

A finite set of trackable objects representing elements of the real-world environment is denoted as follows:

$$TO = \{to_1, to_2, ..., to_t\}, \tag{4.6}$$

where t is the total number of trackables in a CARE environment.

Let $FM = \{fm_1, fm_2, ..., fm_f\}$ be a finite set of fiducial markers, $IM = \{im_1, im_2, ..., im_m\}$ be a finite set of image-based markers, $OM = \{om_1, om_2, ..., om_o\}$ be a finite set of 3D objects as markers, $S = \{s_1, s_2, ..., s_s\}$ be a finite set of surfaces that can be tracked. Formally, TO is a sum of FM, IM, OM, and S. That is,

$$TO = FM \cup IM \cup OM \cup S \tag{4.7}$$

$$\forall to_i \in TO : to_i \in F \vee to_i \in IM \vee to_i \in OM \vee to_i \in S, \quad i \in <1, t> . \tag{4.8}$$

Trackables allow the client device to track its position relative to the real-world environment and to position of tracked objects relative to the device. TO refers to the views registered by a camera indicating the elements of reality to be augmented with objects belonging to CO or DO. Similarly to content and data objects, trackables may be independently provided by different AR service providers. The same trackable object can be reused in multiple AR presentations. For example, a TO representing a movie poster could be used as a tracking source to overlay film trailer onto it, while the same object could be reused to superimpose a 3D-based animation encouraging a user to watch a film in different context.

A trackable may be represented in CARL using the <*Trackable*> element.

Definition 5 - Scenario Object Class

A Scenario Object Class (SOC) – is an abstract class of representations of behavior of AR resources.

Instances of SOC are referred to as scenarios or *so*.

Let $I = \{i_1, i_1, \ldots, i_j\}$ be a finite set of interactions, $C = \{c_1, c_2, \ldots, c_c\}$ be a finite set of object components, and $A = \{a_1, a_2, \ldots, a_a\}$ be a finite set of actions.

A scenario can be denoted as the following triple:

$$so = (I, C, A), \tag{4.9}$$

where:

- I – is a set of interactions assigned to *so*,
- C – is set of object components assigned to *so*,
- A is a set of actions assigned to *so*.

The set of scenarios that can be used to describe behavior of elements of forming a particular CARP is denoted as follows:

$$SO = \{so_1, so_2, ..., so_s\}, \tag{4.10}$$

where s is the total number of scenarios available in a CARE environment.

A scenario may be represented in CARL using the <*Scenario*> element.

Definition 6 - AR Service Provider Class

An AR Service Provider Class – is an abstract class of distributed web service providers delivering AR resources.

Four kinds of AR service providers can be distinguished. Let $P_{CO} = \{p_{CO_1}, p_{CO_2}, ..., p_{CO_w}\}$ be a finite set of content object providers delivering sets of unique *co*, $P_{DO} = \{p_{DO_1}, p_{DO_2}, ..., p_{DO_x}\}$ be a finite set of data object providers delivering sets of unique *do*, $P_{TO} = \{p_{TO_1}, p_{TO_2}, ..., p_{TO_y}\}$ be a finite set of trackable providers delivering sets of unique *to*, and $P_{SO} = \{p_{SO_1}, p_{SO_2}, ..., p_{SO_z}\}$ be a finite set of scenario providers delivering sets of unique *so*.

Formally, content object providers can be denoted as the following function ζ:

$$\zeta : P_{CO} \to CO \Leftrightarrow \zeta \subset P_{CO} \times CO \wedge \forall p_{CO} \in P_{CO}\ \exists! C \subset CO.\ (p_{CO}, C) \in \zeta. \quad (4.11)$$

By analogy, data providers can be described as the function ε:

$$\varepsilon : P_{DO} \to DO \Leftrightarrow \varepsilon \subset P_{DO} \times DO \wedge \forall p_{DO} \in P_{DO}\ \exists! D \subset DO.\ (p_{DO}, D) \in \varepsilon. \quad (4.12)$$

Next, the τ function describes trackable providers:

$$\tau : P_{TO} \to TO \Leftrightarrow \tau \subset P_{TO} \times TO \wedge \forall p_{TO} \in P_{TO}\ \exists! T \subset TO.\ (p_{TO}, T) \in \tau. \quad (4.13)$$

Finally, the σ function describes scenario providers:

$$\sigma : P_{SO} \to SO \Leftrightarrow \sigma \subset P_{SO} \times SO \wedge \forall p_{SO} \in P_{SO}\ \exists! S \subset SO.\ (p_{SO}, S) \in \sigma. \quad (4.14)$$

Definition 7 - CARE User Context

A CARE User Context (CUC) – is a set of properties characterizing the state of a user in a CARE environment.

Based on CUC, AR resources provided by AR service providers, are dynamically discovered, selected, composed to finally form contextual augmented reality presentations. The CARE user context may be explicit (provided by the user) or implicit (automatically retrieved from the user's device). CUC may be composed of the combination of the following elements: user's preferences, time and date, device type (such as smartphone, tablet, or AR glasses), data provided by the GPS and bluetooth modules. Formally, CUC is a pair:

$$CUC = (P, S), \quad (4.15)$$

where:

- P – denotes user's preferences,
- S – denotes user's device state.

Following, user's preferences is an unordered pair:

$$P = \{\alpha, r\}, \quad (4.16)$$

where:

- α – is a set of application domain identifiers $\alpha = \{\alpha_1, \alpha_2, ..., \alpha_n\}$, i.e., categories of AR experiences,
- r – is set of geographical ranges $r = \{r_1, r_2, ..., r_n\}$.

In turn, a device's state is denoted as a 4-tuple:

$$S = (\lambda, \beta, \kappa, \theta), \tag{4.17}$$

where:

- λ – the outdoor location (e.g., the GPS position),
- β – the indoor location (e.g., provided by a beacon device),
- δ – the device type,
- θ – current date and time.

Definition 8 - Contextual Augmented Reality Presentation

A Contextual Augmented Reality Presentation (CARP) – is an augmented reality experience built as a combination of AR resources available in a particular context.

CARP reflects what the user may or wants to experience, taking into account preferences, device capabilities, and spatio-temporal properties, regardless of whether the user is located inside or outside a building. In AR browsers, elements of CARPs are dynamically selected and downloaded from independent service providers, and final CARP is built at runtime of the application – similarly, as web browsers work.

Definition 9 - Description of Contextual Augmented Reality Presentation

---

A Description of Contextual Augmented Reality Presentation (dCARP) – is a description of AR resources constituting a CARP.

Formally, dCARP is the following triple denoted as:

$$dCARP = (\vartheta, \Upsilon, \omega, \psi), \tag{4.18}$$

where $\vartheta, \Upsilon, \omega$, and ψ are multivalued functions responsible for returning subsets of CO, DO, TO, and SO, respectively. Each subset of CO, DO, TO, and SO can be associated with the elements of CUC. Let CX denote a range of user's context (CUC). Let $\vartheta : CUC \rightarrow CO$ denote a multivalued function that assigns subsets of CO to elements of the CUC domain. Function ϑ returns contextual content objects and formally can be presented as follows:

$$\vartheta(CX) = \bigcup_{x \in CX} \{\vartheta(x)\}. \tag{4.19}$$

Let $\Upsilon : CUC \rightarrow DO$ denote a multivalued function that returns contextual business data objects:

$$\Upsilon(B) = \bigcup_{x \in B} \{\Upsilon(x)\} \forall B \subset CUC. \tag{4.20}$$

Let $\omega : CUC \rightarrow T$ denote a multivalued function that returns contextual trackables:

$$\omega(C) = \bigcup_{x \in C} \{\omega(x)\} \forall C \subset CUC. \tag{4.21}$$

In other words, AR resources are assigned to ranges of CUC. A CARE user context plays an essential role in a CARE environment for selecting the best criteria-meeting AR resources. *Semantic Discovery and Matching Method* processes the user's context, while performing SPARQL queries to generate CARP descriptions.

Definition 10 - CARE Knowledge Base

A CARE knowledge base (KB) is a semantic knowledge base containing a set of statements assigning AR resources to particular ranges of context.

The CARE knowledge base is compliant with the *Semantic Augmented Reality Ontology* described in Section 4.4. It contains individuals, their object- and data properties describing characteristics of AR service providers, AR resources, their relationships, and contexts in which these various elements can be composed to build CARPs meeting end-user criteria.

Definition 11 - CARE Environment

Formally, CARE Environment is a 5-tuple denoted as follows:

$$env = (\zeta, \varepsilon, \tau, \sigma, KB, CUC), \tag{4.22}$$

where:

— ζ – are content object providers,
— ε – are data object providers,
— τ – are trackable providers,
— σ – are scenario providers,
— KB – is a CARE knowledge base,
— CUC – is a CARE user context. □

4.3. CARE Architecture

The architecture of distributed augmented reality services in CARE is based on the SOA paradigm (Fig. 4.2). The Service-Oriented Architecture enables building distributed systems that provide application functionality as services to either end user applications or other services. These services are used by end users exploring an AR environment with the use of a contextual AR browser and designers who create elements of CARPs with the use of a 3D software modeler. Splitting CARE functionality between clients and servers permits, on the one hand, to perform complex and extensive computation of semantic processing of distributed AR resources on the server-side, while composition and rendering of CARPs may be dynamically performed in real time on the client-side.

The proposed CARE Architecture consists of two client-side applications – a contextual AR browser and a 3D software modeler. In turn, the server-side is composed of multiple distributed service

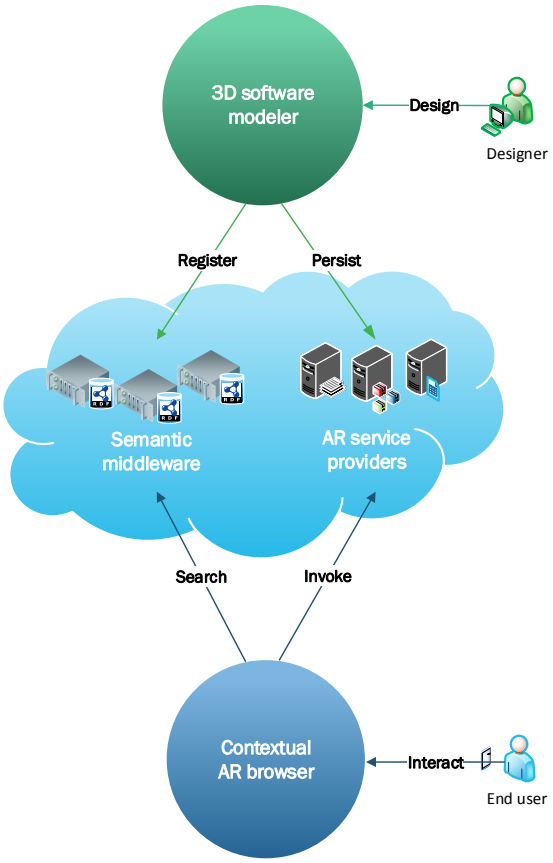


Figure 4.2: The CARE architecture of distributed AR services.

applications – semantic middleware and AR service providers. In the following subsections, a description of each software component is provided.

4.3.1. Semantic middleware

Semantic middleware is a server-based software component providing an API for persisting, discovering, and matching AR service providers and their AR resources based on a user's context. One of the main elements of the middleware is the CARE knowledge base. The semantic middleware constitutes an access layer to the semantic KB via web services – on the one hand, for the designers responsible for creating contextual AR presentations, while on the other hand, for end-users experiencing CARPs with the use of a contextual AR browser component. The semantic middleware provides an API for persisting information about AR service providers, AR resources, and contexts, in which all these elements can be mixed into interactive AR presentations. Moreover, it provides an endpoint for semantic searching and matching AR service providers and AR resources for the contextual AR browser.

4.3.2. AR service providers

AR service providers are independent software components responsible for providing an API for persisting and retrieving AR resources, such as 3D models, images, videos, and sounds. Their main task is to deliver particular AR resources in a given context, to finally achieve a common goal – building interactive AR presentations personalized for end users. AR service providers and AR resources are semantically described within the CARE knowledge base.

4.3.3. Contextual AR browser

A contextual augmented reality browser is a vision-based reusable mobile software component capable of retrieving, processing, and rendering personalized documents describing CARPs (dCARPs).

The novelty lies in the use of the user context that is a key to find relevant distributed AR resources. In the case of a typical web browser, a user manually passes a URI to the address bar to retrieve information resources. Alternatively, a search engine can be used to find interesting resources. In the case of a contextual AR browser, a user interacts with it only by indicating what kind of information he/she is interested in. The device automatically collects the rest of data describing the user context. When the elements of context are collected, the device sends a personalized context to a semantic middleware component that is responsible for matching the user's requirement to available AR service providers and their resources. After that, the semantic middleware responses with a description of CARPs. Once a dCARP has been generated and delivered to the end user, the next task of an AR browser is to interpret it. Then, the application invokes independent distributed AR services to retrieve AR resources that form the CARP. When all AR resources are retrieved and the AR browser is capable of tracking a real-world environment, it starts to render virtual objects on a view registered by the device camera. Last but not least, when the user's context changes and new data come to an AR browser, the same application is capable of preparing a different AR presentation without the need to change the source code.

4.3.4. 3D software modeler

A 3D software modeler is a client-side software component allowing designers to model contextual AR presentations. The tool can be used to design CARPs consisting of multiple content objects, data objects, and trackables. Moreover, the software enables specifying constraints on a user's context in which particular AR resources are available to the end users. Finally, the application communicates with semantic middleware to register AR service providers, AR resources, and context, in which all elements can form CARPs. Also, the software uses the API of AR service providers to persist AR resources forming CARPs.

4.4. Semantic Augmented Reality Ontology

The *Semantic Augmented Reality Ontology* is the central element of the CARE approach as it encompasses the set of concepts and properties describing various aspects of context, resources constituting AR presentations, as well as relations between them [122]. The ontology is used to build ubiquitous dynamic augmented reality environments based on semantically described resources that are contextually provided by AR service providers. SARO is based on the semantic web standards (RDF, RDFS, and OWL 2). SARO consists of six specific sub-ontologies that enable creation of a semantic knowledge base that is compliant with CARE.

Formally, SARO is a 6-tuple denoted as:

$$SARO = (\Sigma, \Delta, B, \Lambda, T, \Theta), \tag{4.23}$$

where:

— Σ - is the AR Service Provider Ontology,
— Δ - is the Domain AR Ontology,
— B - is the Indoor Position Detection Ontology,

— Λ - is the Geography Markup Language Ontology,
— T - is the OWL-Time Ontology,
— Θ - is the Device Type Ontology.

Figure 4.3 visualizes *Semantic Augmented Reality Ontology* as a merged graph.

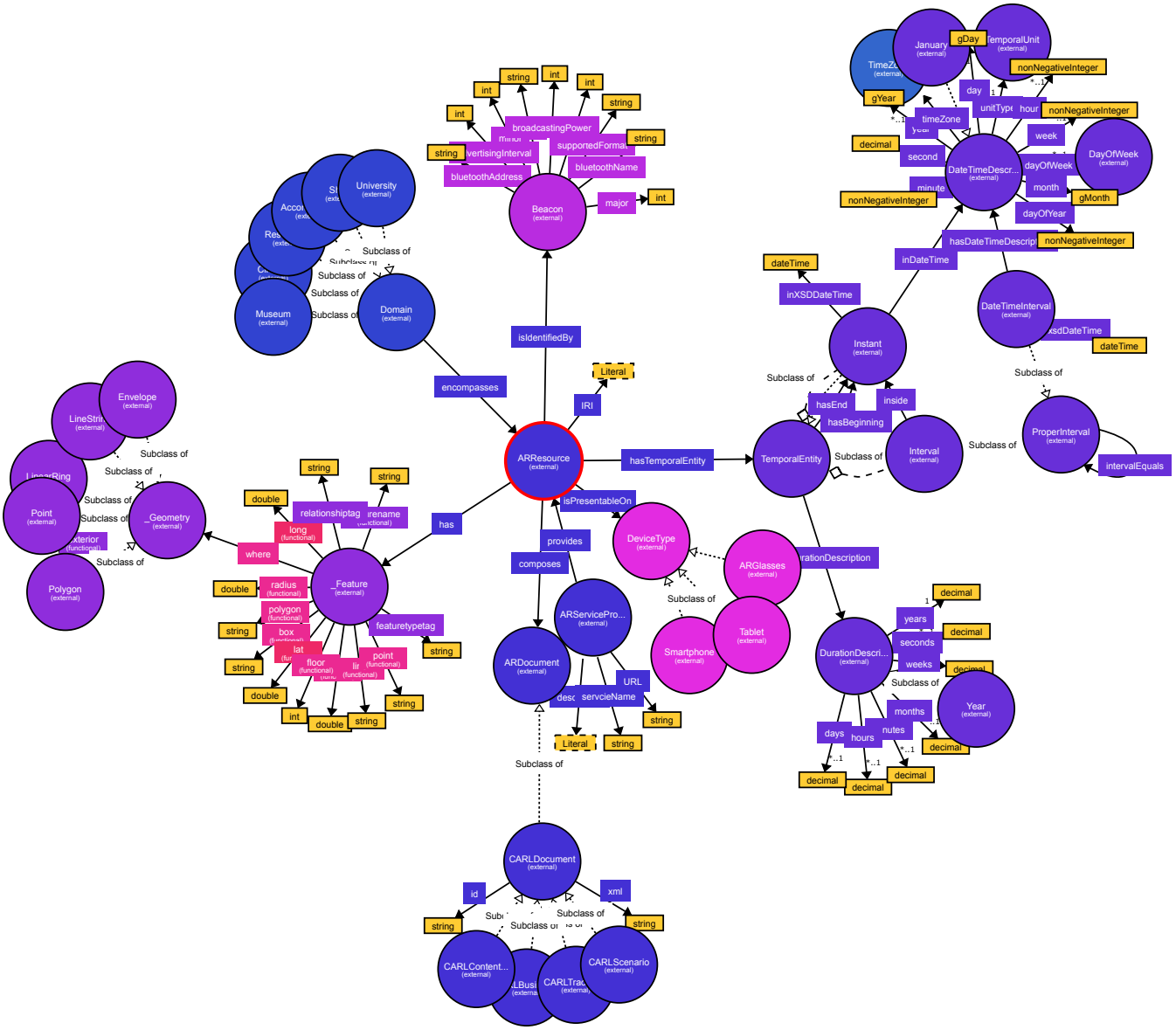


Figure 4.3: Visualization of the SARO ontology.

The following subsections present particular elements of the ontology.

4.4.1. AR Service Provider Ontology

The *AR Service Provider Ontology* (Σ) represents elements of AR services providers responsible for making available diverse AR resources describing the data/content presented in an AR interface. In the ontology, each individual of the *ARResource* class may be associated with a number of instances of *ARDocument* with the object property called *composes*. Instances of particular types of *ARDocument* all together compose dCARPs. SARO is open for extensions, i.e., it defines the *ARDocument* concept

which can be used to build subclasses representing other declarative AR languages, for instance APRIL [81], MRSML [160], KARML [87], ARML [86, 103]. Within this dissertation four classes are proposed: *CARLContentObject*, *CARLDataObject*, *CARLTrackable*, and *CARLScenario* which reflect main elements of the formal model of CARE: CO, DO, TO, and SO.

4.4.2. Domain AR Ontology

The *Domain Augmented Reality Ontology* (Δ) represents a taxonomy of real-world objects that can be augmented with synthetic content through AR presentations (e.g., universities, accommodations, cinemas, shops, etc.). Individuals of these classes may represent particular objects, e.g., buildings of Poznań University of Economics and Business. All domain-specific classes subclass the *Domain* class.

4.4.3. Indoor Position Detection Ontology

The *Indoor Position Detection Ontology* (*B*) enables description of any kind of indoor positioning system. Examples include Bluetooth/BLE, WiFi, RFID, and NFC. Indoor positioning technologies change and develop rapidly. In the presented reference implementation, BLE beacons were used, but this part of the ontology is expected to grow together with the growing availability of technologies and devices.

Beacons are small BLE radio devices transmitting a single signal that other BLE-equipped devices can detect (e.g., smartphones, tablets or wearable AR glasses). Each beacon device broadcasts a globally unique identifier that consists of a combination of letters and numbers transmitted on a regular interval of approximately 1/10th of a second. In the SARO ontology, the individuals of the Beacon class provide context information for AR presentations within an indoor environment (in a case when the user is located within a building). Each individual of the *ARResource* class can be linked with an individual of the *Beacon* class through the object property called *isIdentifiedBy*.

4.4.4. Geography Markup Language Ontology

The *Geography Markup Language Ontology* (Λ) developed by the Open Geospatial Consortium expresses representations of geographical features [101, 102]. GML serves as a modeling tool for geospatial properties for web resources. GML enables describing points, linear rings, rectangles, and polygons as geometric representation properties of geographic features. The GML ontology provides a general feature class (*_Feature*) that can be used to characterize elements of AR presentations as geographic components. To make a statement that a particular *ARResource* has a specific location specified by GML, the *has* object property is used. The specific data property (*where*) associates each feature with one of a limited number of *_Geometry* types, which provide a numerical representation for analysis (e.g., calculating the distance between a user and a particular individual of *ARResource*) and visualization (e.g., showing the possible location of an AR resource on a map). Other data properties describe additional commonly used feature attributes, such as name, elevation, and radius.

4.4.5. OWL-Time Ontology

The OWL-Time ontology (*T*), developed by OGC and W3C, consists of temporal concepts describing the temporal properties of resources in the real world [33]. The ontology defines a vocabulary for expressing statements (facts) about topological relations among *TemporalEntity* individuals, together with information about durations, and about temporal positions including date-time property. OWL-Time

has been chosen to describe a period of time when a particular *ARResource* is available to a user. The *TemporalEntity* class has two subclasses – *Instant* and *Interval*. Instants represent point-like temporal elements that have no interior points. An instant is an interval with zero length, where the beginning and end are the same. Intervals are time periods with a non-zero extent. The class *Interval* has one subclass *ProperInterval* corresponding with the common understanding of intervals, in that the beginning and end are different. The class *ProperInterval* has one subclass – *DateTimeInterval* – whose position and extent may be expressed using a single *DateTimeDescription* or xsd:dateTimeStamp. For instance, an interval can have multiple duration descriptions (e.g., 1 day 4 hours or 28 hours), but can only have one duration.

4.4.6. Device Type Ontology

The *Device Type* (Θ) ontology describes high-level device classes, such as *Smartphone*, *Tablet* and *ARGlasses*. These devices differ in their capabilities – processing power, graphics support, display resolution, field of view, interaction elements and others. In practical applications, to achieve best user experience, AR presentations must be either designed for a specific class of devices or automatically adapted to such a class. The *Device Type* ontology describes for which device type a particular component was intended. The object property *isPresentableOn* is used to link concrete device type to an individual of *ARResource*.

4.5. Semantic Discovery and Matching Method (SDMM)

The CARE mechanism of building contextual AR presentations based on AR resources coming from distributed independent AR service providers relies on *Semantic Discovery and Matching Method* (SDMM). The method is responsible for selecting semantically described AR resources meeting criteria of a user's context. SDMM searches through semantic knowledge base structured with the SARO ontology (presented in the previous section). SDMM is implemented with the use of semantic web standards (RDF, RDFS, OWL 2, and SPARQL) that permit the creation of statements about ubiquitous contextual augmented reality environments.

The Algorithm 1 presents particular steps of the SDMM method that is responsible for composing contextual augmented reality presentations based on multiple distributed AR resources taking into account a user's context. As input data, SDMM takes the elements of CUC that consist of the user's preferences (α and r), indoor or outdoor locations (β or λ), device type (δ), date and time (θ). The α and r elements may be explicitly provided by the user in any AR browser application, or implicitly set based on default values and user's behavior. The β, λ, δ, θ are automatically discovered by the AR browser. As output data, SDMM generates a dCARP description constituting available AR resources within the user's context. The description represents CO, DO, TO, and SO that can be matched to the user's context. The particular elements reflecting AR resources can be based on any declarative AR language that supports building AR presentations, e.g., APRIL, MRSML, KARML, ARML, or CARL.

The SDMM method implements two modes of operation – *firstEnds* and *optimalEnds* – which provide two types of results. In the first mode (*firstEnds*), the SDMM returns the first consistent set of AR resources available within the user's context to form a CARP in the shortest possible time. In the second mode (*optimalEnds*), AR resources constituting a CARP which is the nearest to spatial user's location are returned. In this mode, execution SDMM is significantly more complex and time-consuming.

The first step of the presented algorithm (line 1) is used for searching AR resources within an indoor environment. The method *searchARRIdentifiedByBeacon* takes as the input parameters the following

input : a user's context consisting of: selected individuals of Domain class (α), geographical ranges (r), an indoor location (β), an outdoor location (λ), device type (δ), date and time (θ)
output: a dCARP description

```
1  dCARP ← searchARRIdentifiedByBeacon(β, δ, θ, α);
2  if dCARP ! = null then
3  |  dCARP.generateARDocument();
4  |  return dCARP;
5  end
6  dCARP ← searchGeoARR(λ, α, δ, θ);
7  for i ← 0 to dCARP.length do
8  |  for j ← 0 to dCARP[i].ϑ.length do
9  |  |  ϑ ← dCARP.ϑ[j];
10 |  |  dist ← ϑ.getDistance(λ);
11 |  |  if dist <= r then
12 |  |  |  if isFirstModeSet() then
13 |  |  |  |  break;
14 |  |  |  else
15 |  |  |  |  if dist < bestDist then
16 |  |  |  |  |  best_ϑ ← ϑ ;
17 |  |  |  |  end
18 |  |  |  end
19 |  |  end
20 |  end
21 |  for j ← 0 to dCARP[i].ϒ.length do
22 |  |  ϒ ← dCARP.ϒ[j];
23 |  |  dist ← ϒ.getDistance(λ);
24 |  |  if dist <= r then
25 |  |  |  if isFirstModeSet() then
26 |  |  |  |  break;
27 |  |  |  else
28 |  |  |  |  if dist < bestDist then
29 |  |  |  |  |  best_ϒ ← ϒ ;
30 |  |  |  |  end
31 |  |  |  end
32 |  |  end
33 |  end
34 |  for j ← 0 to dCARP[i].ω.length do
35 |  |  ω ← dCARP.ω[j];
36 |  |  dist ← ω.getDistance(λ);
37 |  |  if dist <= r then
38 |  |  |  if isFirstModeSet() then
39 |  |  |  |  break;
40 |  |  |  else
41 |  |  |  |  if dist < bestDist then
42 |  |  |  |  |  best_ω ← ω ;
43 |  |  |  |  end
44 |  |  |  end
45 |  |  end
46 |  end
47 end
```

```
48 if isFirstOptionSet() then
49 |   dCARP ← new dCARP(ϑ, ϒ, ω);
50 else
51 |   dCARP ← new dCARP(best_ϑ, best_ϒ, best_ω);
52 end
53 dCARP.generateCARLDocs();
54 return dCARP;
```

Algorithm 1: Algorithm of *Semantic Discovery and Matching Method*.

data: indoor location (β), device type (δ), date and time (θ), and selected individuals of the Domain class (α). These data are used to build a SPARQL query for retrieving individuals describing available AR resources within the user's context. In the case when AR resources available in a user's context are found, a description of a new CARP is generated on the basis of the selected individuals (line 3). The method ends by returning a description of CARP consisting of AR resources available in the given indoor spatial context (line 4).

Next – line 6 – the method *searchGeoARR* searches all AR resources associated with a geographical location (λ), chosen domain individuals (α), device type (δ), date and time (θ). Within lines 7-47, geographical filtering of AR resources is carried out in relation to the location of the user. Namely, within lines 8-20, SDMM iterates through all AR resources representing content objects and for each of them the distance to the user is measured. If the distance is less or equal to a given range (line 11) then two options can be considered depending on the SDMM mode of operation. If the first option is set (*isFirstOptionSet*() returns true - line 12), the loop is broken and the method starts to look for contextual data objects (lines 21-33). In other case, the method works as long as it iterates all AR resources in an array to seek the optimal result. Loops presented in the lines 21-33 and 34-46 perform analogously for searching data objects and trackables in an order.

The last part of the algorithm represents creation of a new dCARP description instance (line 49 or 51) on the basis of AR resources localized in a user's geographical surrounding; generating an AR document (line 53); and returning the result (line 54).

4.6. The CARL Language

The CARL language – *Contextual Augmented Reality Language* – is an XML-based AR language that has been designed to support building contextual AR presentations [120]. CARL conceptually reflects main elements of CARE model: CO, DO, TO, SO, and context. The presented language is mainly used to represent dCARP descriptions generated by *Semantic Discovery and Matching Method*. In the following subsections, the main elements of the CARL language are presented.

4.6.1. Content Objects

The first element of an AR environment specification in CARL is the *ContentObjects* element that represents 3D models, images, videos, and sounds to be presented to end users as augmentation of the real-world objects. Each *ContentObject* has a unique *id* that can be used, e.g., to call object actions or to link a concrete content object with a trackable object. Content objects support states. The initial object state is specified by the *initState* attribute. The state can be changed by the use of the *ContentObjectState* command in actions.

The *Resources* element contains information about particular *Components* and *Locations* of component data. Several locations can provide alternative IRIs for different versions of a component meeting particular criteria, e.g., regarding the model complexity or language. Figure 4.4 depicts an example 3D model represented by component with *id* = "*Bars*" (List. 4.1, line 5). Next, the *Actions* element specifies content object-defined actions that can be called on the content object. An action is declared with the *Action* element that has a mandatory *name* attribute and an optional *state* attribute. If the state is specified, the action (if called) will be executed only when the object is in the given state. Within the *Action* element, commands based on the VR–BML [155] language are used. These commands can be used to load, manipulate and animate a component, and change the state of a content object.

Listing 4.1 presents an example declaration of a content object *BookRating1* with a component *Bars* representing user ratings in the form of 3D bars. The *Bars* component has been declared in two versions differing in the level of detail (List. 4.1, lines 6-7). For instance, when a user uses a device with low hardware capabilities or limited network bandwidth, the component version with lower level of detail can be used.

Listing 4.1: An example of a content object declaration.

```
1  <ContentObjects>
2   <ContentObject id="BookRating1" initState="hidden">
3
4    <Resources>
5     <Component id="Bars">
6      <Location detail="low" iri="http://semantic3d.org/cos/bl"/>
7      <Location detail="high" iri="http://semantic3d.org/cos/bh"/>
8     </Component>
9    </Resources>
10
11   <Actions>
12
13    <Action name="show" state="hidden">
14     <SetPosition comp="Bars" x="0" y="0" z="20"/>
15     <SetScale comp="Bars" scale="5"/>
16     <SetOrientation comp="Bars" axis="x" angle="0"/>
17     <Activate comp="Bars" active="true"/>
18     <ObjectState value="shown"/>
19    </Action>
20
21    <Action name="hide" state="shown">
22     <Activate comp="Bars" active="false"/>
23     <ObjectState value="hidden"/>
24    </Action>
25   </Actions>
26  </ContentObject>
27 ...
```

The initial state of the content object is set to the value *hidden*. The *Actions* element is used for grouping actions that can be executed on the declared content object – in this case the *BookRating1* object. The first action named *show* enables setting the position, the scale and the orientation of the *Bars* component and updating the state of the *BookRating1* content object. The second action permits hiding the component and changing the content object state.

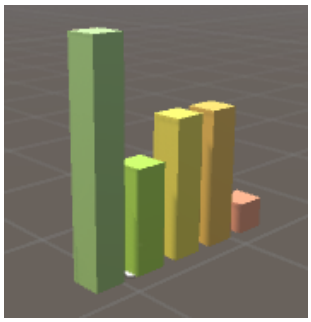


Figure 4.4: An example 3D model that is represented in Listing 4.1 (lines 5–8).

4.6.2. Data Objects

The next element of the CARL language is the *DataObjects* element indicating a section with particular data objects. Each *DataObject* element consists of three parameters: a unique *id*, *isNumerical* taking true/false values indicating whether a data object is numerical or textual, and *iri* that points out a value provided by a data object provider. The value can reflect, e.g., price, duty hour, text information that may change over time. Listing 4.2 presents an example of a declaration of numerical and textual data objects.

Listing 4.2: An example of data objects declaration.

```
1  <DataObjects>
2   <DataObject id="price" isNumerical="true" iri="http://semantic3d.org/rest/price/1"/>
3   <DataObject id="info" isNumerical="false" iri="http://semantic3d.org/rest/info/i1"/>
4   ...
5  </DataObjects>
```

4.6.3. Trackables

The *Trackables* element (Listings 4.3, 4.4, 4.5) groups *Trackable* elements that are used for describing real-world objects that an AR application can detect and track in the 3D space in a given context. The parameter IRI points to binary resource data representing the real-world trackable objects (visual markers).

A *Trackable* element may contain three other elements: *Begin*, *Active* and *End*. The *Begin* section is executed when the trackable is identified in the camera view. It can be used for initializing content objects assigned to the trackable. The *Active* element describes rules of interaction when the trackable object is already detected and tracked by the application. The *End* section is executed when the trackable disappears from the camera view (with a specific delay).

Listing 4.3 presents an example of three *Trackable* elements and three linked *ContentObject* elements that jointly form an AR scene. Each *Trackable* element is associated with an *id* of a *ContentObject* (section *Begin*) to present. This enables displaying appropriate content on the recognized real-world objects. In the presented example, the *BookRating1* content object, from Listing 4.1, was assigned to the *Book1* trackable. Similarly, the content objects *BookRating2* and *BookRating3* are assigned to the trackables *Book2* and *Book3*, respectively. Figure 4.5 depicts three books augmented with three content objects.

Listing 4.3: Three trackables with assigned content objects.

```
1  <Trackables iri="http://semantic3d.org/books.dat">
```

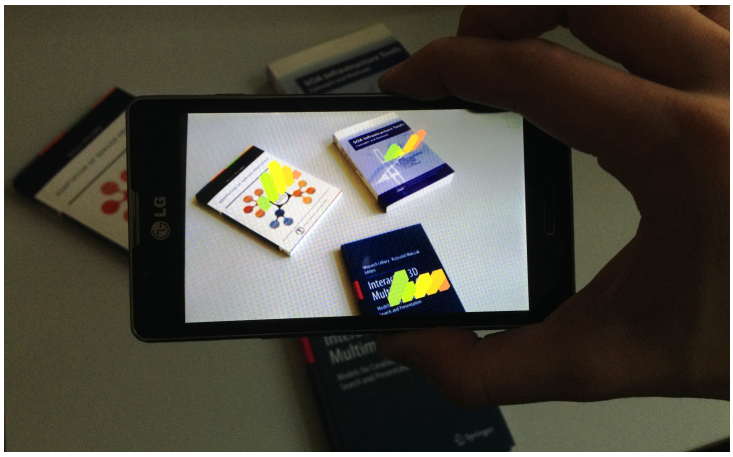


Figure 4.5: Augmentation of three books reflecting the CARL description presented in Listing 4.3.

```
<Trackable id="Book1">
 <Begin>
  <ObjectBegin select="Book1Rating1 "/>
 </Begin>
</Trackable>

<Trackable id="Book2">
 <Begin>
  <ObjectBegin id="BookRating2"/>
 </Begin>
</Trackable>

<Trackable id="Book3">
 <Begin>
  <ObjectBegin id="BookRating3"/>
 </Begin>
</Trackable> ...
```

Another case of an AR scene modelled with the CARL language is presented in Listing 4.4. In this scene, two content objects *BookRating1* and *BookPrice1* are assigned to a single trackable element – with the id: *Book1*. Figure 4.8 (right) presents an example book augmentation with the AR scene declared in Listing 4.4.

Listing 4.4: A trackable with two content objects assigned.

```
<Trackables iri="http://semantic3d.org/books.dat">
 <Trackable id="Book1">
  <Begin>
   <ObjectBegin id="BookRating1"/>
   <ObjectBegin id="BookPrice1"/>
  </Begin>
 </Trackable>
</Trackables>
```

4.6.4. Sectors

To enable conditional presentation of visual and aural content depending on the relative position of the camera and a real-world object, the *Sector* element was introduced in CARL [121]. The *Sector*

element declares an active volumetric sector in the 3D space relative to a trackable object, which can trigger actions when the camera enters (the *In* element) or leaves (the *Out* element) the sector boundaries. It contains information about what actions should be triggered when the camera enters or leaves the sector. An example action is fading in or out a sound when camera enters or leaves a sector. The actions that should be triggered when the camera enters the sector are specified within the *In* element. In turn, the 'leaving' actions are specified within the *Out* element.

The sector boundaries are described by four ranges. The first range (*min*_α, *max*_α) indicates pitch (elevation) rotation between the camera's position and the reference plane of the trackable object. The second range (*min*_β, *max*_β) indicates yaw (heading) rotation between the camera position and the tracked real object. The third range (*minDist, maxDist*) specifies the minimal and the maximal distance from the camera to the center of the trackable object. This distance is calculated on the basis of the length of the translation vector. The last range (*minHeight, maxHeight*) describes the minimal and the maximal height between the camera and the trackable object.

Listing 4.5: An example of a trackable with sectors.

```
<Trackables iri="http://semantic3d.org/books.dat">
 <Trackable id="Book1">

  <Begin>
   <ObjectBegin id="BookRating"/>
   <ObjectBegin id="BookPrice"/>
  </Begin>

  <Active>
   <Sector id="Rating" minInc="0" maxInc="90"
    minRot="0" maxRot="360" minDist="0" maxDist="2">

    <In>
     <ObjectAction id="BookRating" action="show"/>
    </In>

    <Out>
     <ObjectAction id="BookRating" action="hide"/>
    </Out>

   </Sector>

   <Sector id="Price" minInc="0" maxInc="90"
    minRot="0" maxRot="360" minDist="0" maxDist="1">

    <In>
     <ObjectAction id="BookPrice" action="show"/>
    </In>

    <Out>
     <ObjectAction id="BookPrice" action="hide"/>
    </Out>

   </Sector>
  </Active>
  ...
  <End> ... </End>
```

```
38   </Trackable>
39  </Trackables>
```

In Listing 4.5, two sectors *Rating* and *Price* are defined for the trackable *Book1*. Within these sectors, different *In* and *Out* actions are specified. When the camera enters the *Rating* sector, the action *show* is called on the *BookRating* content object. On leaving the sector, the action *hide* is executed on the *BookRating* content object. With the given sector definition, the book rating information will be displayed when the user's device is within the distance of 2 meters from the book cover.

When the camera moves closer to the book and enters the *Price* sector (defined as a hemisphere with the radius of 1 meter), the application triggers the action *show* on the *BookPrice* content object to display the price of the book in addition to the already displayed rating. The *hide* action declared within the *Out* element, hides the book price information when the camera is moved away from the book, out of the *Price* sector.

Figure 4.6 schematically presents sectors defined for the trackable in this example. An AR browser calculates pitch, yaw, height, as well as the distance between the camera and the trackable object. By monitoring these parameters, the application can call appropriate actions when entering and leaving particular sectors.

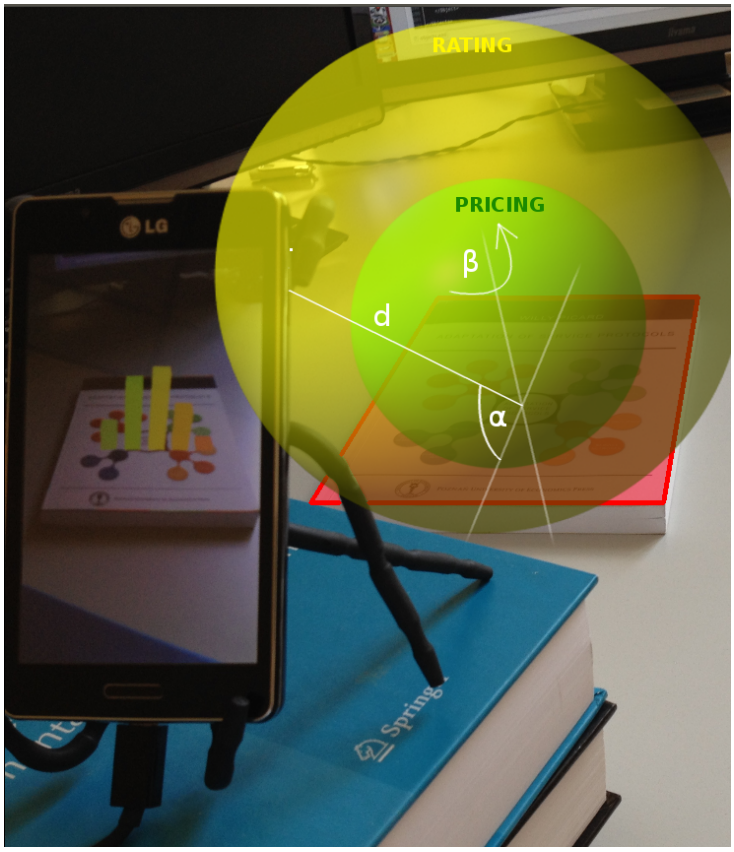


Figure 4.6: Sectors defined by the distance (d), inclination (α) and rotation (β) between the camera and a trackable object.

4.6.5. Spatial sound in CARL

3D sound is a valuable cue for navigation in AR environments. The use of spatial sound in AR environments can be a significant factor in searching and navigating for hidden objects within indoor

environments. To enable the use of audio content in a CARE environment, the CARL language provides audio elements [116]. Moreover, to control auditory objects depending on the relative position of the camera and a real-world object, the *Sector* element, presented in section 4.6.4, can be used.

An example auditory object, presented in Listing 4.6, has been described as a *ContentObject* element consisting of *Resources* as well as *Actions* elements. The *Resources* element points to locations of the same audio element with different levels of quality. The *Actions* element describes actions that can be called on the content object. In this case, actions are responsible for controlling audio (e.g., starting/stopping audio, setting right/left channel's volume, and setting the sound's looping). These actions are triggered by an AR browser, depending on where the camera is situated in 3D space.

Listing 4.6: A description of content object representing the lion's roaring sound.

```
1  <ContentObjects>
2   <ContentObject id="lionFX">
3    <Resources>
4     <Component id="http://.../res/lionFX">
5      <Location details="high" iri=".../lionFX"/>
6     </Component>
7    </Resources>
8
9    <Actions>
10    <Action name="startRoaring">
11     <SetLooping>true</SetLooping>
12     <SetVolume left="5" right="5"/>
13     <Play/>
14    </Action>
15
16    <Action name="stopRoaring">
17     <Stop/>
18    </Action>
19
20    <Action name="setVolume($v)">
21     <SetVolume left="$v" right="$v"/>
22    </Action>
23
24    ...
25    <Action name="moreOnRight($v)">
26     <SetVolume left="($v - 30)" right="$v"/>
27    </Action>
28    ...
29    <Action name="roar">
30     <SetVolume left="99" right="99"/>
31    </Action>
32   </Actions>
33  </ContentObject>
34 ...
```

Listing 4.6 presents the *ContentObject* with an *id* = *lionFX* representing lion's roaring. The sound's resource is identified by *iri=http://.../res/lionFX*. Six actions are declared within the *lionFX* content object that can be called by an AR browser. The first action – *startRoaring* – is responsible for setting up and starting the sound effect (lines 10–14). At first, it sets the sound's looping to true to repeat it without any gap between its end and start (line 11). Next, it sets the volume level to five units on *left* and *right* channels (line 12). In the end, it starts to play the sound effect with the *play* command (line 13).

The next action – *stopRoaring* – stops audio content representing lion's roaring sound (lines 16–18). To increase or decrease the volume, the *setVolume* action can be used, which takes as a parameter *$v* the level of volume. Next, the *moreOnRight* action increases volume level to *$v* on the right channel and to ($v - 30) on the left channel, giving a sense that the sound's source comes from the right side of a user's camera (lines 25–27). The last action – *roar* – sets the volume up to both channels to the maximum value indicating that the user is close to the source of sound (lines 32–34). These actions will be used when a user's camera approaches the source of sound to give a notion of depth perception effect.

Listing 4.7: Sectors responsible for starting and controlling sounds.

```
1  <Trackable id="landscape">
2
3  <Begin>
4   <ObjectBegin id="lionFX"/>
5   <ObjectBegin id="lion3D"/>
6   ...
7  </Begin>
8
9  <Active>
10 <Sector name="startToPlay"
11        minAlpha="0" maxAlpha="180"
12        minBeta="0" maxBeta="360"
13        minDistance="0" maxDistance="1500"
14        minHeight="0" maxHeight="1500">
15  <In>
16   <ObjectAction id="lionFX" action="startRoaring"/>
17   <ObjectAction id="pigFX" action="startSquealng"/>
18   <ObjectAction id="pigFX" action="startNeighing"/>
19  </In>
20
21  <Out>
22   <ObjectAction id="lionFX" action="stopRoaring"/>
23   ...
24  </Out>
25 </Sector>
26
27 <Sector name="lionVolume10"
28        minAlpha="0" maxAlpha="60"
29        minBeta="35" maxBeta="50"
30        minDistance="1400" maxDistance="1500"
31        minHeight="1212" maxHeight="1300">
32  <In>
33   <ObjectAction id="lionFX" action="setVolume(10)"/>
34  </In>
35 </Sector>
36
37 <Sector name="lionVolume20"
38        minAlpha="0" maxAlpha="60"
39        minBeta="35" maxBeta="50"
40        minDistance="1300" maxDistance="1400"
41        minHeight="1126" maxHeight="1212">
42  <In>
43   <ObjectAction id="lionFX" action="setVolume(20)"/>
44  </In>
45 </Sector>
```

```
46 ...
```

Listing 4.7 presents a shortened description of the trackable object with declared sectors for starting and controlling sounds. Lines 3-7 present audio as well as visual objects that are associated with the *landscape* object. The first sector *startToPlay* starts sound effects of animals (lines 10–25). When a user's camera enters the sector boundaries, then the AR browser triggers *startRoaring*, *startSqueaking*, and *startNeighing* actions using *ObjectAction* elements declared within the *In* element (lines 15–19). In turn, when the camera leaves the *startToPlay* sector, the application calls actions declared within the *Out* element - in this case sounds of animals will be stopped (lines 21–24). The next two sectors - *lionVolume10* and *lionVolume20* - are used when a user approaches to the lion's sound source (lines 27–45). In this case, decreasing the distance causes the volume of lion's sound to be turned up.

Listing 4.8: Combining spatial sound with a visual content object.

```
1  <Sector name="lionOnRight"
2          minAlpha="0" maxAlpha="60"
3          minBeta="15" maxBeta="35"
4          minDistance="0" maxDistance="600"
5          minHeight="0" maxHeight="600">
6   <In>
7    <ObjectAction id="horseFX" action="setVolume(70)"/>
8    <ObjectAction id="lionFX" action="moreOnRight(50)"/>
9    <ObjectAction id="pigFX" action="littleOnLeft(40)"/>
10   <ObjectAction id="horse3D" action="showHorse"/>
11   ...
12  </In>
13  ...
14 </Sector>
15
16 <Sector name="oppositeToLion"
17         minAlpha="0" maxAlpha="60"
18         minBeta="35" maxBeta="50"
19         minDistance="0" maxDistance="500"
20         minHeight="0" maxHeight="500">
21  <In>
22   <ObjectAction id="lionFX" action="roar"/>
23   <ObjectAction id="lion3D" action="showLion"/>
24   ...
25  </In>
26  ...
27 </Sector>
28 ...
```

Listing 4.8 describes sectors that trigger actions for combining spatial sound with visual content. The *lionOnRight* sector declares actions that will be called when a camera points to the center of a trackable object from the right side and also the distance between the camera and the trackable object is small (lines 1–12). The first three actions give an effect that the neighing (line 7) is the most audible sound. The roaring sound is also audible, but in this case only on the right channel (line 8). The squealing sound is little audible on the left channel (line 9), but it does not play an important role in this sector. This construction gives a clue that the lion object can be located on the right side of the horse. The last action (*showHorse* – line 10), called on content object with the *id=horse3D*, shows a 3D model representing the horse object.

The last sector – *oppositeToLion* – contains actions that are called when a camera points to the right side of the trackable object (lines 16–27). The first action – *roar* (line 22) – is responsible for turning the roaring sound up. This action is called on content object with the *id=lionFX*. The next action – *showLion* (line 23) – triggered on content object with *id=lion3D*, shows a 3D model of a lion.

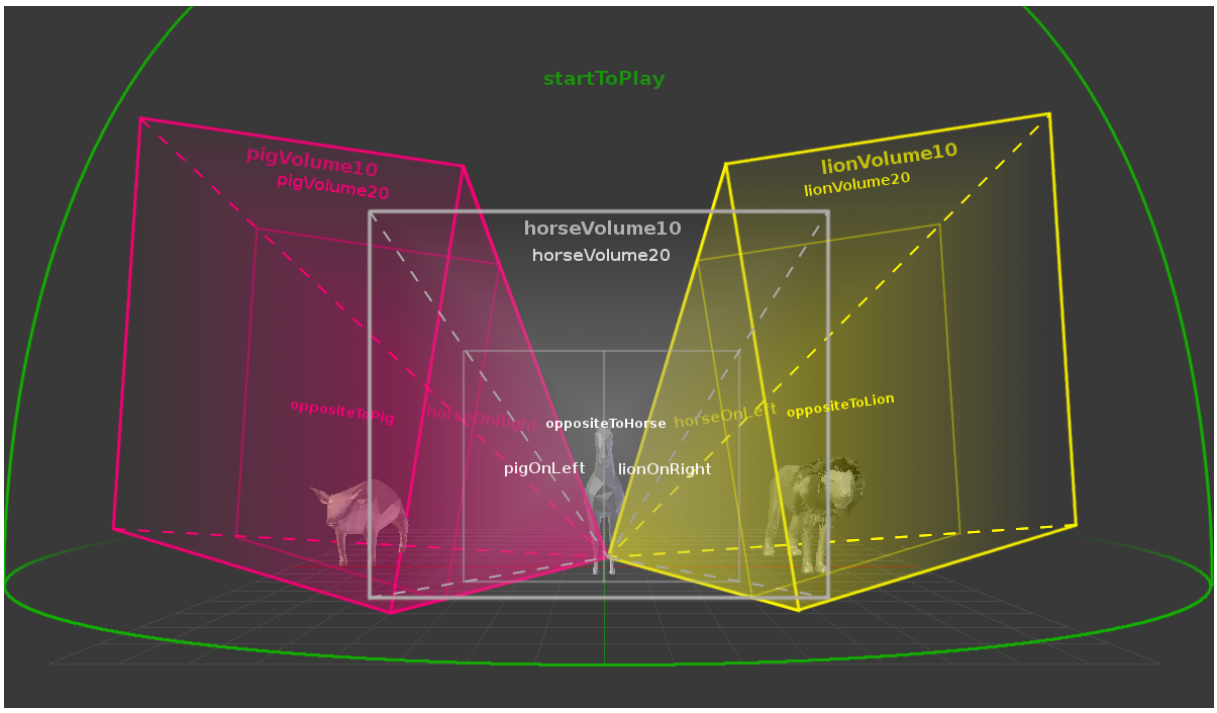


Figure 4.7: Visualization of CARE audio sectors.

4.6.6. Scenarios

CARL specifies scenarios in which different types of interactions can be declared in order to trigger CARL-defined or content object-defined actions. CARL-defined actions are *Move*, *Rotate*, and *Scale*.

A scenario can distinguish interactions between content objects and trackables using the parameter *type*. Interaction with content objects enables activation and manipulation of objects. Interaction with trackables can be used, e.g., to display objects associated with the tracked markers. In order to recognize on which AR resources a particular action supposed to be called, the parameter *target* is used. This parameter takes an id of an AR resource as a value.

CARL specifies the following interactions *SingleTap*, *DoubleTap*, *LongPress*, *Pan*, and *Pinch*. Listing 4.9 presents an example of a scenario specification, in which a single (lines 4–6) and a double tap (lines 8–10), long press (lines 12–14), pan (lines 16–18) and pinch (lines 20–22) gestures trigger specific actions.

Listing 4.9: An example of scenario specification.

```
1  <Scenarios>
2   <Scenario id="Scenario1">
3
4    <SingleTap target="Book1" type="Trackable">
5     <Trigger action="show(*)"/>
6    </SingleTap>
7
```

```
8    <DoubleTap target="Book1" type="Trackable" >
9     <Trigger action="hide(*)"/>
10   </DoubleTap>
11
12   <LongPress target="*" type="ContentObject">
13    <Move x="0.5" y="0.35" z="1"/>
14   </LongPress>
15
16   <Pan target="*" type="ContentObject" >
17    <Rotate x="0.1" y="0.1" z="2" rotation="0.5"/>
18   </Pan>
19
20   <Pinch target="*" type="ContentObject" >
21    <Scale x="0.5" y="0.5" z="0.5"/>
22   </Pinch>
23
24  </Scenario>
25 </Scenarios>
```

4.6.7. Context

Elements of CARL can be described with the CARE user context (CUC). The *CUC* elements groups *Context* elements. Each *Context* specifies a *key* and a *value*, which jointly define a constraint facet. Values of these parameters reflect data describing user's context. Listing 4.10 presents an example declaration of the user context encoded in CARL.

Listing 4.10: An example of the CARE user context declaration.

```
1 <CUC>
2   <Context key="Domain" value="Cinema"/>
3   <Context key="DeviceType" value="ARGlasses"/>
4   <Context key="starts" value="2017-12-10T09:31:10.6"/>
5   <Context key="ends" value="2017-12-31T23:59:59.0"/>
6   ...
7 </CUC>
```

4.6.8. Dynamism in CARL

In order to enable loose coupling with AR resources, the identifiers can have a form semantic expressions. In this case, values CARL elements' parameters are dynamically generated on the basis of semantic queries. A description of a contextual AR presentation can be generated using a CARL template. In this case, values of CARL elements' parameters are dependent on a user context and they are dynamically generated on the basis of semantic queries. The final description of a CARP is unknown to an AR system implementing CARL until a system queries a semantic knowledge base in order to fill-in a template with semantic data, and finally to generate a dCARP.

Listing 4.11 depicts an exemplary CARL-template (lines 8–18) that is described with the user's context (lines 3–6). The presented template links each content object with a concrete trackable. SDMM generates as many such pairs as a semantic query returns results. To meet this requirement, the *Foreach* element has been used (line 8).

Listing 4.11: An example of the semantic template declaration.

```
...
 <CUC>
  <Context key="uuid" value="d0d3fa86-ca76-45ec-9bd9-6af444fdbfc7"/>
  <Context key="domain" value="Restaurant"/>
  ...
 </CUC>
 ...
 <Foreach>
  <ContentObject id="?co_id">
  ...
  <Trackables iri="?iri">
   <Trackable id="?to_id">
    <Begin>
     <ObjectBegin id="$co_id"/>
    </Begin>
   </Trackable>
  </Trackables>
 </Foreach>
...
```

Having a particular CARL-template on the middleware-side, SDMM can run a semantic query against a CARE knowledge base in order to generate a final description of a contextual AR presentation. Listing 4.12 presents an exemplary SPARQL query run by SDMM to generate a dCARP.

Listing 4.12: An example of the semantic query supporting the CARL-template from Listing 4.11.

```
SELECT ?pco ?co ?co_id ?pto ?to ?to_id ?iri ?beacon ?uuid

WHERE {
 ?pco saro:isIdentifiedBy ?beacon .
 ?pco saro:provides ?co .
 ?co saro:id ?co_id .
 ?pto saro:isIdentifiedBy ?beacon .
 ?pto saro:provides ?to .
 ?pto saro:IRI ?iri .
 ?to saro:id ?to_id .
 <http://www.semantic3d.org/dr/daro#Restaurant> daro:encompasses ?to .
 <http://www.semantic3d.org/dr/daro#Restaurant> daro:encompasses ?co .
  ?beacon beacon:uuid  "d0d3fa86-ca76-45ec-9bd9-6af444fdbfc7"^^xsd:normalizedString .
    ...
}
```

4.6.9. Application example

Figure 4.8 shows an example of using an AR browser application in a bookstore. Based on a user's preferences and the position context (a particular bookstore), the application retrieves trackables from AR trackable service provider belonging to owners of the bookstore. Retrieved trackables represents books that are potentially interesting for the user. The context information greatly reduces the number of visual markers that need to be identified and tracked by the application in the real time. This – in turn – reduces the power consumption and improves stability of tracking.

When a registered book cover becomes visible to the camera, the AR browser starts tracking the book. With the CARL trackable specification provided in Listing 4.5 (p. 68), nothing is displayed on

the book until the user enters the *Rating* sector (yellow hemispherical area depicted in Figure 4.6, p. 69). Then, the rating information is retrieved from a publisher web service and is displayed on the book cover in the form of colored 3D bars – from the highest rates to the lowest (Figure 4.6 left, p. 69). This gives the user a possibility to compare ratings of different books (cf. Figure 4.5, p. 67). Also, when the user moves the camera closer, the book price information is shown from the bookstore web service and is displayed on the book cover next to the rating. The book price depends on the bookstore and may also depend on the user. The price information together with the rating may help the user to undertake the buying decision.

Figure 4.8: An example of CARL application – Bookstore AR service.

4.7. Summary

In this chapter, the main contribution of this dissertation – the *Contextual Augmented Reality Environment* (CARE) approach – has been described. The overall concept of CARE, including the formal model and the main elements of CARE have been presented.

At first, the architecture of distributed AR services with particular software entities, including two client applications (the contextual AR browser and the 3D software modeler) and multiple server-based applications (a semantic middleware and AR service providers), has been described. Application of SOA and semantic matching and discovery mechanism enables to dynamically combine diverse and distributed AR resources into consistent AR presentations in a contextual manner. Performing semantic computation in the CARE middleware eases the client applications from time-consuming operations and enables to dynamically build contextual AR presentations without the need of changing the source code of AR browsers.

Next, the *Semantic Augmented Reality Ontology* that enables to model independent AR service providers, resources, and contexts and link all these elements to build a ubiquitous dynamic AR environment, has been discussed. Using the presented ontology, it is possible to create numerous semantic knowledge bases that can be applied in many application domains, such as tourism, marketing, public transport information systems, etc.

After that, the *Semantic Discovery and Matching Method* that is responsible for selecting semantically described data meeting criteria of a user's context has been described. SDMM searches through semantic knowledge base structured with the SARO ontology and creates a description of a contextual AR presentation.

Finally, a new high-level language, called CARL – *Contextual Augmented Reality Language*, used for encoding the generated contextual AR presentations has been presented. CARL descriptions are interpreted by client-side AR browsers to provide users with contextualized AR experiences.

5. Implementation of CARE

This chapter provides an overview of the reference implementation of the CARE system. First, the system architecture including data flow diagrams between its main components is presented. Further, the client-side and server-side components are described. Particular sections include descriptions of the following software components: Unity3D-based plugin, called *CARE Modeler*, which is used for modeling contextual augmented reality environments; *BrowsAR* – an Android-based augmented reality browser enabling to experience contextual AR presentations; Java EE-based application server *Semantic Augmented Reality Middleware* – responsible for providing semantic search functionality; and multiple RESTful web services offering AR resources that can be dynamically composed to build diverse-contextualized augmented reality presentations. After that, real-world examples of CARE environments are presented. Finally, conclusions to the chapter are provided.

5.1. Architecture and data flow

Figure 5.1 depicts the overall architecture of the implemented CARE system prototype and data flow visualizing the communication between the main elements of the system. The green arrows present activities of a CARE designer who performs semantic modeling process of AR environments, while the blue arrows depict actions taken by end-users while exploring the AR environment. On the server side, the orange arrows show Jena framework [15] operations, as well as executions of SPARQL queries, run against the semantic knowledge base.

5.1.1. Semantic modeling of AR environments

The semantic modeling process of AR environments begins with the use of the *CARE Modeler* tool. The tool is used by a designer who does not require to have technical programming skills. When *CARE Modeler* starts, as a first step, it automatically performs an HTTP GET operation (1) to retrieve the list of semantically described categories (2) of AR service providers. After that, the tool performs the second HTTP GET operation (3) to obtain semantically described device types (4). Further, these two semantic collections are supplied to the *CARE Modeler* graphical user interface to semantically describe AR resources.

When the process of downloading categories semantics is completed without HTTP errors, the designer can start the work. The designer is responsible for arranging AR scenes consisting of trackables, content objects, and data objects. Then, each of AR resources is separately sent via HTTP POST method to (possibly independent) AR service providers, as follows: trackables provider (5), content objects provider (5'), and data provider (5"). After it is published, the same AR resource can be reused in different use cases dependent on the user's context. Listing 5.1 presents an example request sent by *CARE Modeler* to a content object provider.

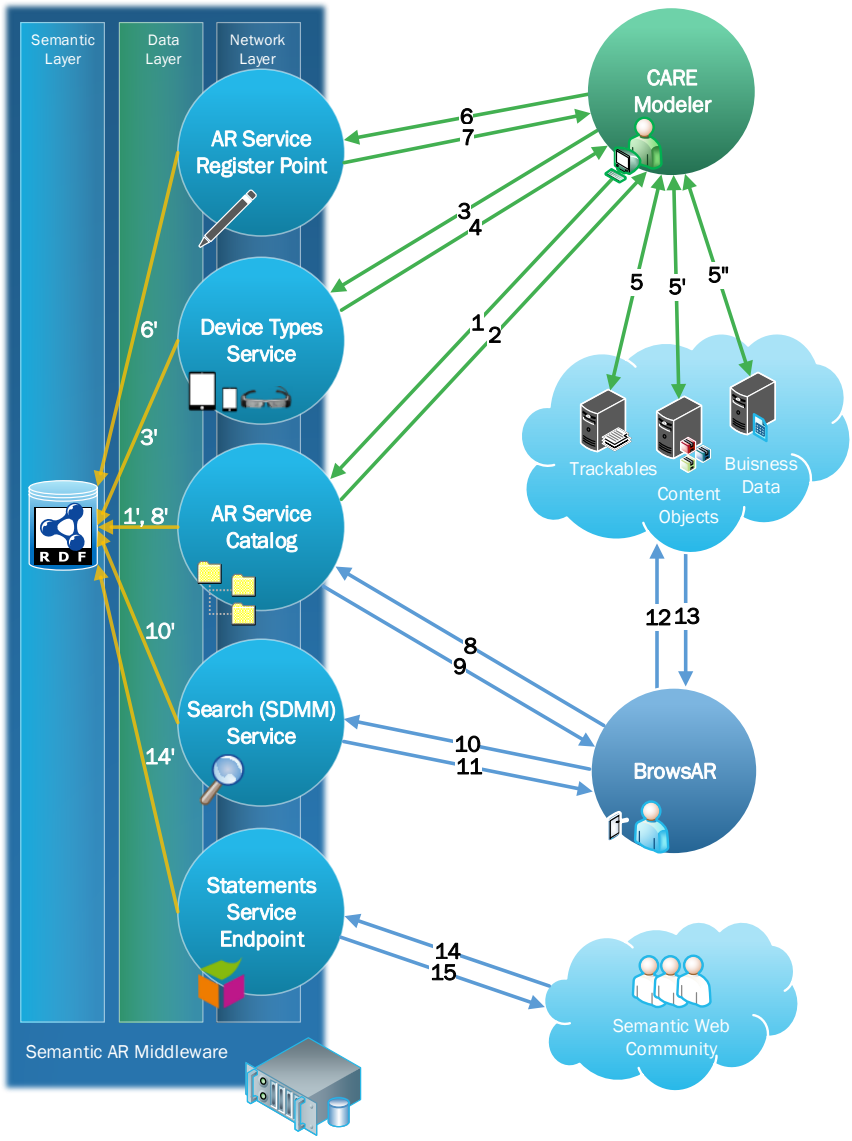


Figure 5.1: Data flow diagram of the CARE system.

Listing 5.1: A shortened example of a request sent to a content object provider (5').

```
{
"name": "sculpture_carl",
"type": "obj",
"contextPath": "carl/models/3d/",
"model64":")ZyBjdWJlMl9jYXJsCnYgLTAuNDMzNDc2NyAtMC42OTgwMTQ3IDAuMjczNjMwMQp2IDAu
MTQwMDE3NiAtMC4xMTMwMjg3IDAuODQ3MTI0Mwp2IC0wLjg0NzEyNDMgMC4xMTMwMjg3IC0wLjE0MDAxN
zUKdiAtMC4yNzM2MyAwLjY5ODAxNDcgMC40MzM0NzY3CnYgLTAuMTQwMDE3NiAwLjExMzAyODcgLTAuOD
Q3MTI0Mwp2IDAuNDMzNDc2NyAwLjY5ODAxNDcgLTAuMjczNjMwMQp2IDAuMjcNjMgLTAuNjk4MDE0NyAt
NjQ3NiAwLjgxMTA0MzQgLTAu==",
"textures64": ["iVBORw0KGgoAAAANSUhEUgAAAQAAAAEACAYAAABccqhmAAAABmJLR0QA/wD/AP+gv
AAAsTAAALEwEAmpwYAAAAB3RJTUUH4QIGER8KfgxA/AAAABl0RVh0Q29tbWVudABDcmVhdGVkIHdpdGgg
R0lNUFeBDhcAACAASURBVHjabL1ZryRJkqX3iaqa+XKXWDMckiBADIEBfyl/AF/4NG8ES
WFe0V15+8Rqs8Hd//++4u3vGArjvMNK7B4BcMjkJwzBwOOwjCVyW103tnGArjMNK7B4BcMjkJwzBwOOwjOCZyKVtUu==",]
}
```

The designer need to know addresses of each AR service provider to enter data into the *CARE Modeler* interface to correctly dispatch elements of a CARP. Technically, *CARE Modeler* encodes data such as 3D models, textures, trackables objects to the base64 representation [69] and sends it to an appropriate AR service provider. As a response, each AR service provider sends information back whether the data have been correctly persisted. *CARE Modeler* collects the HTTP responses and notifies the designer about the resource publishing process. If no errors have been detected and the AR resources have been published correctly on AR service providers, the designer can start the next task – context modeling.

In the context modeling phase, the designer specifies the context in which a combination of the above-mentioned resources can be used to form AR presentations. To model the context, not only lists retrieved in steps (2) and (4) are used, but also data describing user's geographical position, characteristics of a beacon, and time-frames, in which the AR resources can be used to compose an AR presentation. Finally, when the whole process of modeling is finished, the designer registers a description that links the designed AR resources (available within AR service providers) with the modeled context to an *AR Service Register Point* (6). The service responses with HTTP status code (7). The designer can iteratively repeat steps of the semantic modeling process to build complex contextual AR environments.

5.1.2. Exploration of AR environments

End users can experience AR presentations with the use of the *BrowsAR* mobile application. When the mobile application starts, it sends a request (8) to get a list of semantically described categories of AR resources – as indicated by arrow (1). After *BrowsAR* loads information about categories, the end user can set the context by selecting interesting categories manually. Moreover, the end user can provide a distance range, which will be used to discover resources within the context range based on the geographical location. At the same time, *BrowsAR* automatically collects data from GPS and Bluetooth devices to recognize outdoor/indoor position. Moreover, *BrowsAR* harvests data from the device operating system to check what kind of device type is used by the end user. Additionally, to determine whether AR glasses are used, Vuforia *isSeeThru()* method is tested. In the case, when *BrowsAR* collected all needed data, the application dispatches a representation of context to *Search Service* (10).

Listing 5.2: An example of an end-user context encoded with JSON.

```
{
   "location":{
     "longitude":-74.009464,
     "latitude":40.706276
   },
   "beacon": {
       "uuid":"d0d3fa86-ca76-45ec-9bd9-6af444fdbfc7",
       "major":3113,
       "minor":22979
   },
   "range":100,
   "domain":"http://www.semanticweb.org/dr/2016/daro#Museum",
   "deviceType":"Smartphone",
   "userDateTime":"2017-12-20T09:30:10.5"
}
```

An example description of end-user context encoded as a JSON object, which is sent to *Search Service*, is presented in Listing 5.2. In the presented case, the context consists of the geographical

location (lines 2-5); outdoor location connected with a specific beacon (lines 6-10); range of searching boundary set to 100 meters (line 11); semantic category *Museum* (line 12); and device type *Smartphone* (line 13). The context was read on 20th December 2017 at 09:30:10 (line 14).

Listing 5.3: An example of response sent by *Search Service* to *BrowsAR* (11).

```
1  [{"poi":{"location":{"longitude":-74.009481,"latitude":40.706281},
2   "featureName":"sculpture",
3   "feature":"http://www.semantic3d.org/dr/SARO#_f_sculpture",
4   "distanceToUser":7.79},
5   "contentObjectXML":"
6    <?xml version=\"1.0\" encoding=\"UTF-8\"?>
7    <ContentObjects>
8     <ContentObject id=\"sculpture_carl\" initialState=\"hidden\">
9      <Resources>
10      <Component id=\"c5\">
11       <Location details=\"low\" uri=\"http://www.semantic3d.org/app/sculpture_carl\"/>
12      </Component>
13     </Resources>
14     <Actions>
15      <Action name=\"init\" state=\"hidden\">
16       <SetPosition component=\"c5\" x=\"0\" y=\"0\" z=\"0\"/>
17       <SetOrientation component=\"c5\" axis=\"x\" angle=\"90\"/>
18       <SetOrientation component=\"c5\ axis=\"y\" angle=\"0\"/>
19       <SetScale component=\"c5\" scale=\"40\"/>
20       <Activate active=\"true\" component=\"c5\">
21        <ObjectState value=\"shown\"/>
22       </Activate>
23      </Action>
24     </Actions>
25    </ContentObject>
26   </ContentObjects>",
27  "trackableXML":"<?xml version=\"1.0\" encoding=\"UTF-8\"?>
28   <Trackables>
29    <Trackable id=\"musemuFX=\"http://www.track3d.org/app/carl/trackables/museumFX.dat\"
30     uriXML=\"http://www.track3d.org/app/carl/trackables/museumFX.xml \"
31     thumbnail=\"http://www.track3d.org/app/carl/trackables/museumFX.jpg \">
32     <Begin>
33      <ObjectBegin id=\"sculpture_carl\"/>
34     </Begin>
35    </Trackable>
36   </Trackables>",
37   "dataObjectXML":null, "interfaceXML":null
38 },
39 {"poi":{"location":{"longitude":-74.008781,"latitude":40.705919},
40 "featureName":"atm",
41 "feature":"http://www.semantic3d.org/dr/SARO#_f_atm",
42 "distanceToUser":67.96},
43 "contentObjectXML":null,"trackableXML":null,"dataObjectXML":null,"scenarioXML":null}]
```

As a response (11), *BrowsAR* retrieves an array of the dCARP descriptions meeting criteria of end-user context. Listing 5.3 shows an example response dispatched to the browser application. The presented example consists of two JSON objects: the first element (lines 1-38) and the second (lines 39-43). What is interesting in this example is that the first element is supplemented with the CARL description pointing to AR resources available at distributed AR service providers (lines 11 and 29-31).

However, the second JSON object does not contain such information (line 43) due to the fact that the distance between end-user and the location of AR resources assigned to the second CARP presentation (line 42) is greater than the distance to the first CARP (line 4). Thus, SDDM does not generate CARL descriptions for subsequent JSON objects due to unnecessary data redundancy (line 43), but it only provides additional data that can be, e.g., overlaid on a map in order to give the end user a hint where other CARPs can be found. Further, the AR browser performs HTTP GET methods (12) to get best criteria-meeting resources (13) of the AR presentation from distributed AR service providers.

5.1.3. Providing an endpoint to external programs

In accordance with the main principle of the semantic web, which is sharing and reusing data, *Semantic Augmented Reality Middleware* additionally implements an open interface called *Statements Service Endpoint*. The interface can be used by researchers and developers to retrieve knowledge about contextual AR environments. In such a way, the semantic web community can adapt and integrate SARO concepts with other semantic-based systems.

The bottom part of Figure 5.1 presents an interface enabling external programs to perform HTTP GET method (14) to retrieve semantic statements representing CARE environments in the form of *subject, predicate, object*. As a response, *Statements Service Endpoint* produces an XML description that represents semantic triples (15) of the CARE knowledge base. Figure 5.2 depicts an example response of *Statements Service Endpoint* retrieved through a web browser.

5.2. Client-side CARE components

5.2.1. The *CARE Modeler* application

CARE Modeler is a tool that was implemented to help designers and developers to model augmented reality presentations based on distributed AR resources with the contextual approach. The tool can be used to model AR presentations consisting of multiple content objects and trackables. The software enables also to specify context in which particular resources can be available to end-users. The tool conforms to the SARO ontology. The application is implemented in C# as an editor extension plugin to the Unity3D game engine [139]. Additionally, the plugin uses data provided by the Vuforia Unity extension [110].

Figures 5.3 and 5.4 present the graphical user interface of the CARE Modeler. The meaning of the particular parts of the interface (annotated with numbers in Fig. 5.4a) are presented in the following subsections.

Modeling contextual AR presentations

With the use of Unity IDE a developer can use standard transform tools, such as move, rotate, and scale, to position content objects. The transform properties of the content objects will be further mapped into CARL content object descriptions. The content objects are positioned with the respect to a particular representation of the view of reality (trackables). These objects are provided by the Vuforia Unity extension as a marker database. The name of the database should be provided in the *Database Name* field of the *CARE Modeler* interface (Fig. 5.4a, part (2)). Moreover, the developer can optionally prepare a thumbnail representing a view of the reality which can be augmented in *BrowsAR* . Further, this thumbnail will be used by an AR browser to give the end-user a hint, where he/she should point a mobile device to experience an AR presentation in the particular context. After positioning content objects, the

developer can preview the AR presentation using the Vuforia Unity extension by clicking on the play button (as depicted in Fig. 5.5).

Next, the *Optional CARL Tools* section (Fig. 5.4a, part 5) can be used to generate and preview CARL descriptions of the AR presentation on the basis of the Unity3D scene. Figure 5.4b presents unfolded CARL tools element, where the developer can preview an automatically generated CARL description. Optionally, the designer can modify the CARL description to provide other elements of CARL such as scenarios or sectors – assuming that the designer knows specification of the CARL language.

Modeling context

The *User's Context* section (Figures 5.4a, part (6) and 5.6) is divided into four parts: *Beacon*, *Location*, *Device type*, and *Date and time*. Here, the developer can set up particular parts of the context, in which the designed AR resources will be available to end-users.

Figure 5.7 depicts properties of a beacon device in which *uuid*, *major*, and *minor* fields can be set up. These properties can be used to localize AR in indoor environments. For instance, in the case when a user is located in a building and the AR browser receives signals from a particular beacon device, let say characterized with the following features:

1. *uuid*: d0d3fa86-ca76-45ec-9bd9-6af4789f73ce;
2. *major*: 58906;

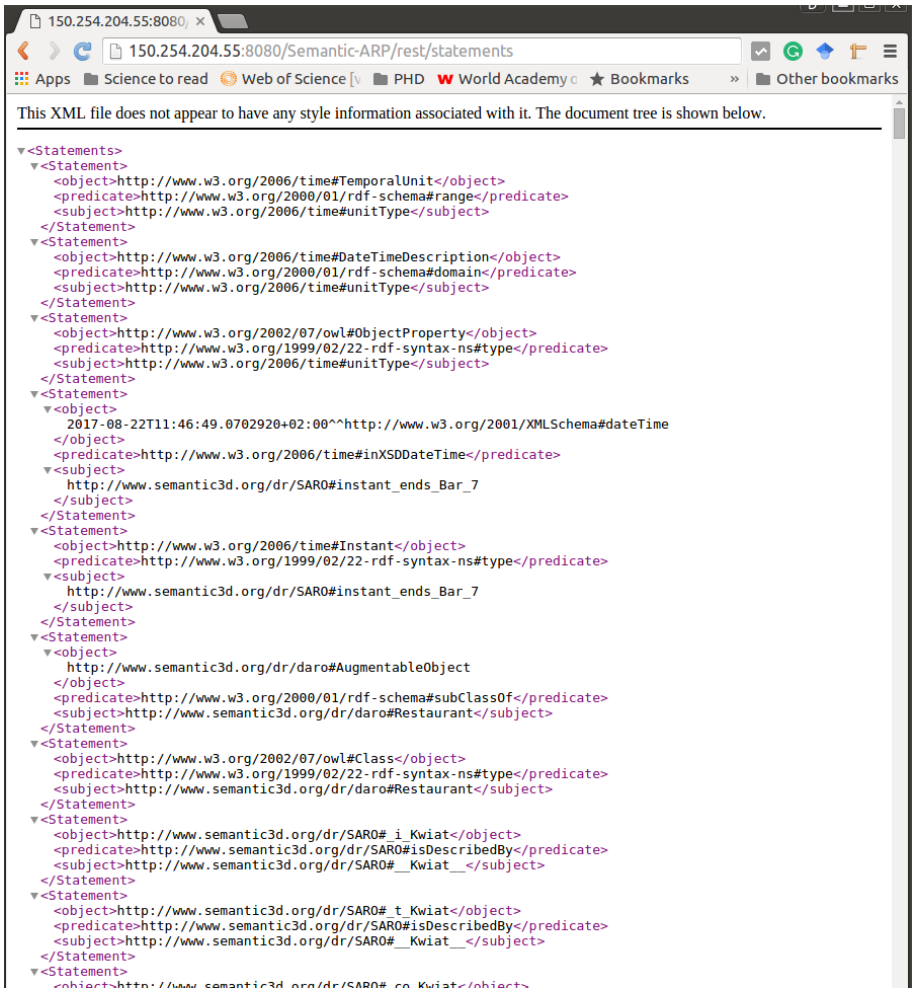


Figure 5.2: Semantic statements retrieved from *Statements Service Endpoint*.

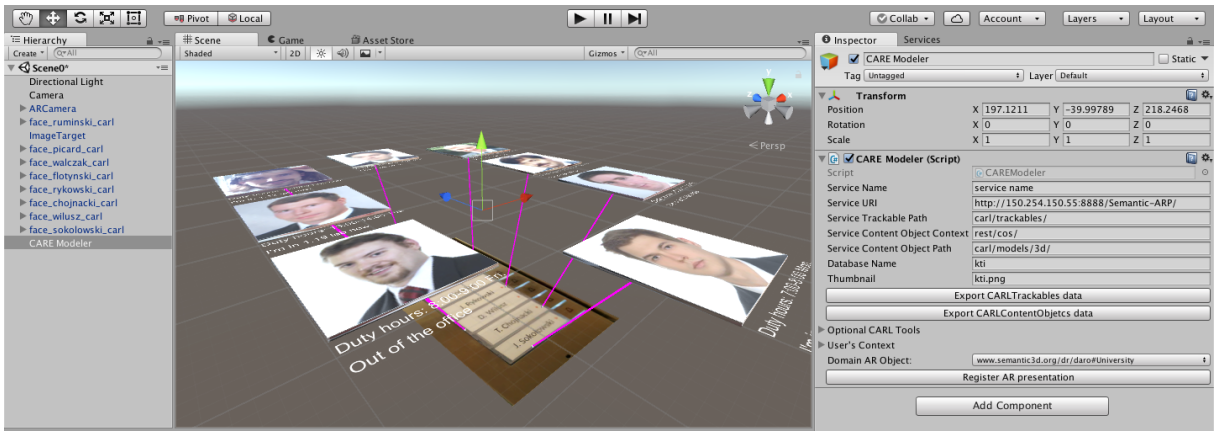
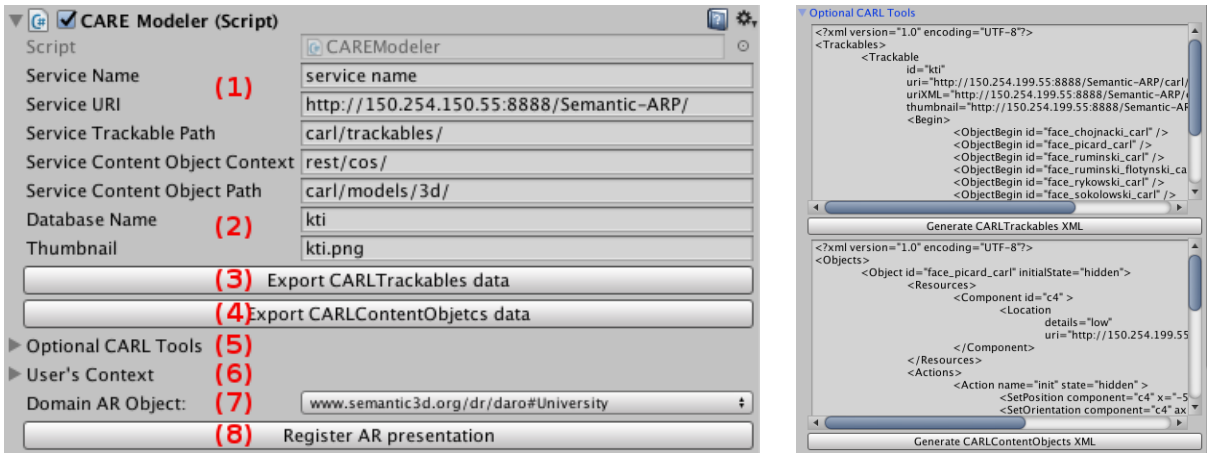


Figure 5.3: Modeling an AR presentation with the CARE Modeler tool.



(a) (b)

Figure 5.4: Primary GUI of *CARE Modeler* (a). Unfolded *Optional CARL Tools* section (b).

3. *minor*: 10758;

an AR presentation associated with context identified by this beacon will be dynamically composed in run-time, and – in the end – presented on the end-user's device, when the user turns his/her device into a particular view of the reality. An example of RDF representation of a beacon device encoded with the RDF-XML notation is presented in Listing 5.4. The CARE middleware creates an individual of Beacon class consisting of the SARO namespace, prefix *_b*, major number, four first letters of uuid, and generated random number.

Listing 5.4: An example of RDF beacon representation.

```
1  <rdf:RDF
2     xmlns:SARO="http://www.semantic3d.org/dr/SARO#"
3     ...
4     >
5
6     <SARO:isIdentifiedBy>
7       <Beacon rdf:about="http://www.semantic3d.org/dr/SARO#b_58906_d0d39802">
8         <uuid>d0d3fa86-ca76-45ec-9bd9-6af4789f73ce</uuid>
9         <major rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
10        >58906</major>
```

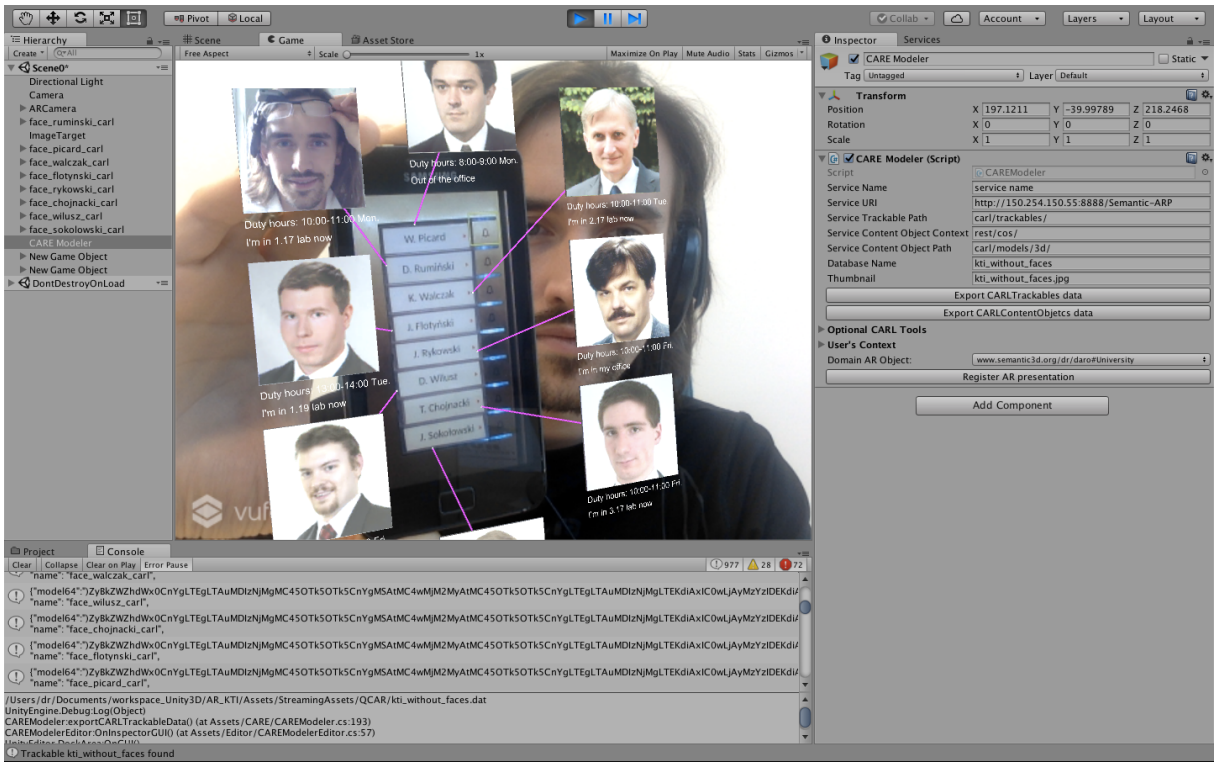


Figure 5.5: Previewing the AR presentation using the Vuforia Unity extension.

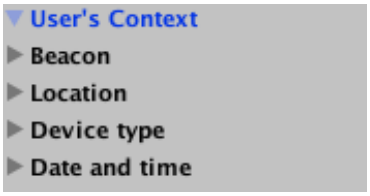


Figure 5.6: Unfolded *User's context* section.

```
        <minor rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >10758</minor>
      </Beacon>
    </SARO:isIdentifiedBy>
  ...
```

The next panel – *Location* – presented in Figure 5.8, enables setting outdoor location properties, such as: GML feature name, longitude, and latitude, which describe a spatial outdoor context associated with AR resources. Information about particular AR presentations that can be built within the end-user's surrounding will be available in the AR browser. The developer can combine outdoor location with indoor location data when modeling a context for an AR presentation. For instance, the AR browser

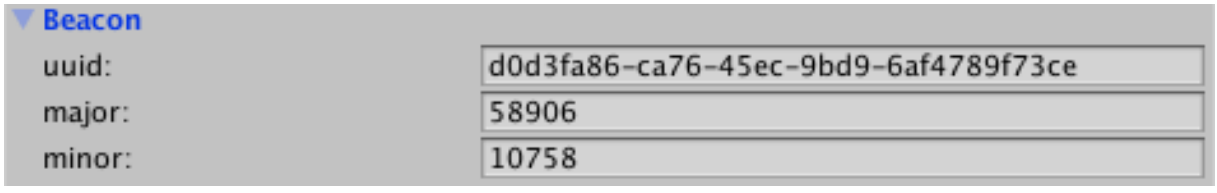


Figure 5.7: Setting up beacon properties in *CARE Modeler*.

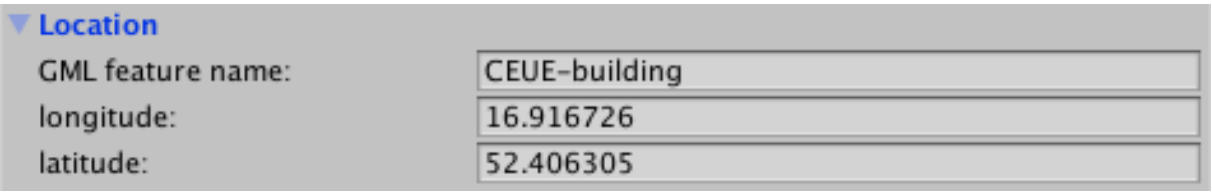


Figure 5.8: Setting up outdoor location properties in *CARE Modeler*.

Listing 5.5: An example of RDF outdoor location representation.

```
<rdf:RDF
   xmlns:SARO="http://www.semantic3d.org/dr/SARO#"
   ...
   >

   <SARO:has>
     <gml:_Feature rdf:about="http://www.semantic3d.org/dr/SARO#_f_CEUE-building">
       <gml:featurename>CEUE-building</gml:featurename>
       <geo:where>
         <gml:Point rdf:about=
         "http://www.semantic3d.org/dr/SARO#loc_point_CEUE-building">
           <geo2003:long rdf:datatype="http://www.w3.org/2001/XMLSchema#double"
           >16.916726</geo2003:long>
           <geo2003:lat rdf:datatype="http://www.w3.org/2001/XMLSchema#double"
           >52.406305</geo2003:lat>
         </gml:Point>
       </geo:where>
     </gml:_Feature>
   </SARO:has>
...
```

application can show on a map positions of places in which different AR resources are associated with beacon devices. In that case, when the user enters the building and GPS signal is lost, indoor locations of AR resources can be distinguished with beacons to build appropriate AR presentations inside the building. Listing 5.5 presents an example of a semantically described spatial context, in which an AR presentation may be available to end-users.

Figure 5.9 presents the *Device type* drop-down list consisting of semantically described device types for which AR resources are aimed. The same AR presentation may be displayed differently on various devices. Also, devices may differ in their ability to handle different types of objects. Hence, in some cases it is necessary to model presentations with different properties for various device types to give end-users the best user experience when presenting AR presentations. Elements of the *Device type* drop-down list are dynamically downloaded before the *CARE Modeler* tool is available to use by the designer. These data come from the CARE service responsible for providing supported device types.

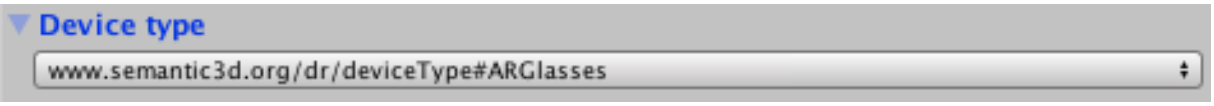


Figure 5.9: Selecting a device type for an AR presentation in *CARE Modeler*.

Last but not least, the *Date and time* panel (Fig. 5.10) enables the developer to specify beginning and expiration dates and times of the AR presentation, optionally marked with a particular time zone offset. The CARE Modeler tool automatically generates default values, however, the developer can change the setting.

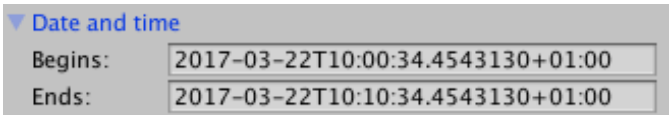


Figure 5.10: Setting validity period of AR resources in *CARE Modeler*.

The lexical representation of beginning and expiration dates and times consists of sequences of characters of the following form: *YYYY-MM-DD'T'hh:mm:ss('.' s+)(zzzzzz)*, according to the ISO 8601 standard [67], where:

- *YYYY* indicates the year,
- *MM* indicates the month,
- *DD* indicates the day,
- *T* is a separator indicating that time-of-day follows,
- *hh* indicates the hour,
- *mm* indicates the minute,
- *ss* indicates the second,
- *'.' s+* (if present) represents the fractional seconds,
- *zzzzzz* (if present) represents the timezone.

Figure 5.10 presents the beginning and expiration values differing by ten minutes. The semantic representation of these temporal properties are presented in Listing 5.6.

Listing 5.6: An example of temporal properties described in RDF-XML.

```
...
  <SARO:hasTemporalEntity>
    <time:TemporalEntity rdf:about="http://www.semantic3d.org/dr/SARO#TE_example">
      <time:hasEnd>
        <time:Instant rdf:about="http://www.semantic3d.org/dr/SARO#instant_ends_example">
          <time:inXSDDateTime rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime"
          >2017-03-22T10:10:34.4543130+01:00</time:inXSDDateTime>
        </time:Instant>
      </time:hasEnd>
      <time:hasBeginning>
        <time:Instant rdf:about="http://www.semantic3d.org/dr/SARO#instant_starts_example">
          <time:inXSDDateTime rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime"
          >2017-03-22T10:00:34.4543130+01:00</time:inXSDDateTime>
        </time:Instant>
      </time:hasBeginning>
    </time:TemporalEntity>
  </SARO:hasTemporalEntity>
...
```

Registering AR resources

The annotation (1) of Figure 5.4a (p. 84) points to AR service provider properties that are used to persist AR resources. The first five fields declare properties that are used for storing data coming from the Unity3D IDE. When these values are set, a developer can export representations of visual markers used for augmentation and their graphical thumbnails (used for guiding users to appropriate real-world elements) to the external service – by pressing the *Export CARLTrackable data* button (3). Then, virtual

objects constituting the AR scene can be exported by pressing the *Export CARLContentObjects data* button (4).

The next section – *Domain AR object* (7) – assigns AR resources to a particular domain-specific category of AR experiences. In this case, these AR resources will be available in search results in case of choosing the *University* keyword preference.

Finally, the button *Register AR presentation* (8) is used for registration of AR resources with the specific context in the CARE catalog REST service. Technically, it sends all provided and generated data using the HTTP POST method to the server. Then, these data are stored within the CARE knowledge base.

5.2.2. The *BrowsAR* application

BrowsAR – which stands for "Browser for Augmented Reality" – is an Android-based client application that is responsible for communication with *Semantic Augmented Reality Middleware* and multiple AR service providers, dynamic composition, and real-time rendering of AR presentations based on the contextual approach. *BrowsAR* is built on top of the OpenGL ES library [72] that allows rendering diverse content and data objects. To recognize and track planar images and 3D objects in real time, the Vuforia computer vision library is used [110]. The *BrowsAR* application is based on the REST architectural paradigm [47] and is able to retrieve diverse resources from distributed AR service providers to compose AR presentations encoded in the CARL language. The application has been implemented in Java and runs on the Android platform. With the use of *BrowsAR*, an end-user is able to experience multiple AR presentations that are dynamically built at runtime of the application. Moreover, end-user can control *BrowsAR* with voice commands.

BrowsAR architecture

The multi-layered architecture of *BrowsAR* meets requirements of the Model-View-Controller (MVC) design pattern [50]. The application consists of four layers: *Network Layer* and *Data Layer* (model), *Presentation Layer* (view), and *Context Layer* (controller). Figure 5.11 illustrates the four-layered architecture of *BrowsAR*, in which each rounded rectangle presents a programming library or an Android module that was used to implement the browser. The particular layers are described below.

Network Layer (NL) is responsible for communication with *Semantic Augmented Reality Middleware* and multiple AR service providers. The core element behind the NL is the CARL library, which deals with asynchronous HTTP operations and processing of results without interrupting the UI thread of *BrowsAR*, while end-user context changes and new data are being retrieved. Moreover, the CARL library provides the *CARLFactory* class, which conforms to the Factory Method design pattern [51]. This class interprets dCARP descriptions and transforms them into Java CARL objects, additionally altering these instances with resources retrieved from AR service providers. A *CARLFactory* instance produces *CARLContentObject*, *CARLBusinessData*, *CARLTrackables*, and *CARLScenarios* objects that are ready for further processing by the Context and Presentation layers. The next element of Network Layer is Google play-services-map library. In the implemented prototype, version 9.6.1 of the library is used to support the map component. The library provides access to the Google Maps service [59] and permits to customize information about nearby AR presentations displayed on a map. Last but not least, the *Speech2Text Processor* module is responsible for accessing a speech recognition service to transform end-user voice into textual commands. These commands are further transferred via the Data Layer to

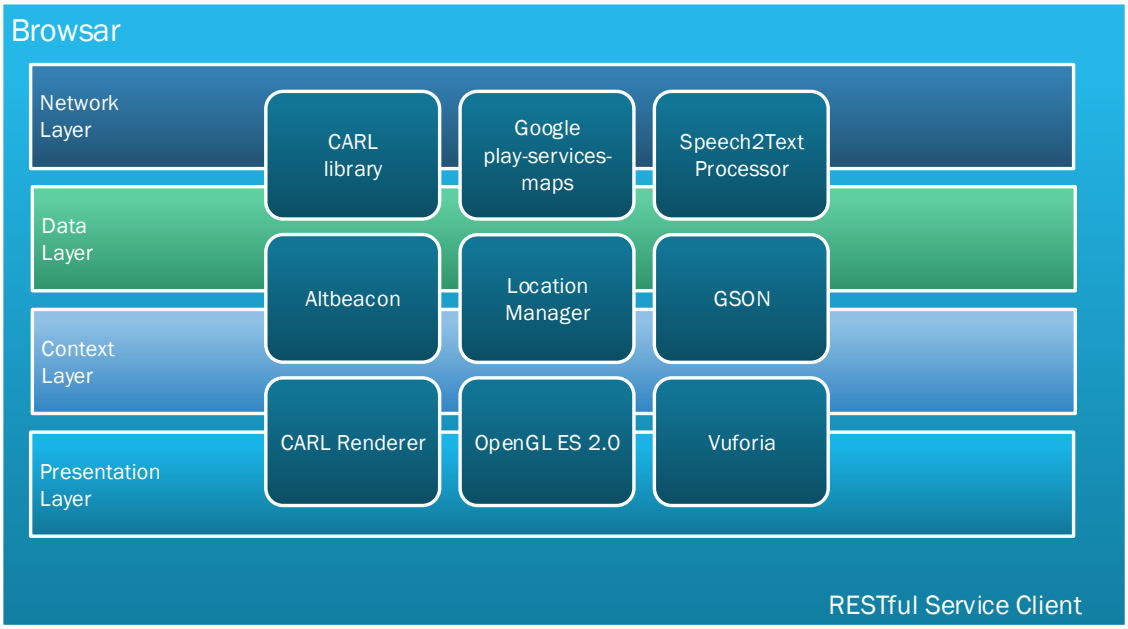


Figure 5.11: The architecture of the *BrowsAR* application.

the Context Layer to analyze and, in the end, to control application with the use of the end-user voice.

Data Layer (DL) constitutes a bridge between the Network and Context layers, i.e., the DL is responsible for handling data retrieved from the NL and transferring it to the CL. Also, the Data Layer collects data from various devices, such as: Bluetooth (via Altbeacon that provides APIs for getting notifications when beacons appear or disappear), GPS (via Android Location Manager to obtain geographical position of the end user), and from the Android operating system and the Vuforia library to retrieve knowledge about the type of device that is used by the end user. To notify objects related to Context Layer when new data come, the DL, as well as the NL, extensively use the Observer design pattern [51]. The Data Layer converts arbitrarily complex Java Objects into their JSON representations and vice versa with the use of the GSON library [55].

Context Layer (CL) is the decision-making point of the application. The foundation of this layer is an instance of the *ContextController* class that is responsible for processing notifications coming from the Network and the Data layers, which have impact on the end-user's context. Based on the received low-level messages, the instance of *ContextController* continually measures whether the context changes, and triggers appropriate actions that influence elements of the DL – by preparing requests to be sent to *Semantic Augmented Reality Middleware*, as well as, elements of the Presentation Layer – by seamlessly altering low-level renderers, without interrupting the UI thread of *BrowsAR*. Furthermore, the Context Layer is also responsible for providing notifications to related software components that "listen" to changes in the context, for instance, by dispatching messages to the the graphical user interface or just to logging mechanisms. Finally, the CL interprets commands received by the *Speech2Text Processor* module and takes actions that are responsible for controlling the application with the use of voice commands.

Figure 5.12: The GUI of *BrowsAR*.

Presentation Layer (PL) is responsible for rendering AR presentations based on the CARL objects provided by an instance of the *CARLFactory* class. In order to implement the AR visualization objective, the PL uses three core libraries as follows: CARL Renderer, OpenGL ES 2.0, and Vuforia. CARL Renderer consists of three independent classes that implement GLSurfaceView.Renderer interface. These classes are responsible for making low-level OpenGL ES calls to render 3D models encoded in the following formats: obj [140], 3ds [144], and md2 [65]. Information, where to position and how to orient particular 3D models, is provided by Vuforia. This library provides an image registration capability which produces a camera projection and model view matrices related to the recognized and tracked marker. These two matrices are further processed by OpenGL to properly transform virtual objects. The Presentation Layer also includes declarative elements of user interface encoded within Android XML files that correspond to the Android View classes and subclasses. This approach enables separation of the graphical user interface elements from the application logic code.

Graphical user interface of *BrowsAR*

Figure 5.12 depicts the graphical user interface of *BrowsAR*. The image is additionally marked with red-numbered annotations of the particular parts of the GUI. Number (1) presents an optional logger tool that is responsible for printing *BrowsAR* logs thus helping a developer to debug low-level messages in the case a mobile device is not connected to a computer with the Android Studio environment [54]. Next, number (2) presents the thumbnail view showing end-users a hint where the camera should be pointed to track a marker and experience the AR presentation. Number (3) depicts the real world object that is registered and tracked by the camera. This object is overlaid with virtual content objects provided by an AR service provider. These 3D objects have been earlier exported by a developer using the *CARE Modeler* tool. Number (4) shows the button that triggers the speech recognition function. With this feature, end-users can control the application with voice commands. After pressing the voice command button, *BrowsAR* plays gentle "beep" sound and listens for one second for a voice command. A description of voice commands is provided in subsection 5.2.2. Number (5) then presents the button used for showing a map on which blue pin icons are marked representing points of interest (POIs) of

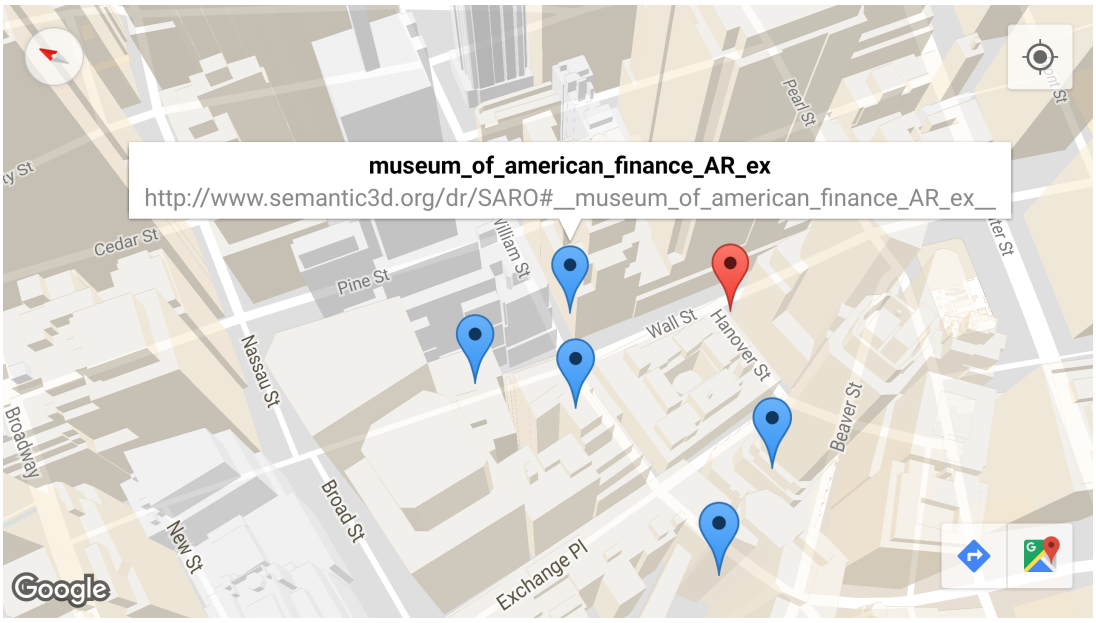


Figure 5.13: The map activity presenting the positions of end-user and five semantically described AR presentations.

possible AR presentations that can be accessed within the range of the end-user context. Additionally, the current geographical position of the end user is labeled as a red pinpoint. The map is configured to hide unnecessary and distracting Google's business points of interest using the map styling feature [56]. Figure 5.13 illustrates a map activity on which the positions of end-user and five semantically described AR presentations are drawn.

Number (6) presents the button that, after the pressing, triggers a function requesting *Semantic Augmented Reality Middleware* for dCARP descriptions. When this button changes color from yellow to green, it means that end-user context has just changed and the button is enabled for an action. In that case, when *Semantic Augmented Reality Middleware* responds with a non-empty CARL document, *BrowsAR* automatically calls AR service providers to retrieve resources constituting CARPs. When the process of downloading AR resources is finalized, *BrowsAR* changes color of the button (7) from yellow to green, and – in addition – notifies the end user with a textual information that new AR resources have been downloaded and there is a possibility to reload the AR scene. Finally, number (7) presents the button for manually reloading OpenGL renderers with new 3D models and updating the Vuforia tracker mechanism to track new markers. The last annotation – number (8) – depicts semantic categories provided by *AR Service Catalog*.

What is worth to mention is that *BrowsAR* can be setup to fully and automatically work without manual user interactions. For instance, the application can autonomously call remote methods of *Semantic Augmented Reality Middleware* and AR service providers and reload new AR resources in the case, the end-user context changes. However, this configuration by default is turned off due to the fact it causes battery drain of the mobile device.

Voice commands

BrowsAR distinguishes six voice commands as follows:

1. *screen on* – displays the logger tool;
2. *screen off* – hides the logger tool;

3. *map on* – opens a map view with annotated geographical position of the end user and possible AR presentations meeting criteria of context (see Fig. 5.13);
4. *dispatch context* – dispatches end-user context to the *Search Service*; in case of getting non-empty response, *BrowsAR* calls remote methods of AR service providers to retrieve AR resources needed to compose CARPs;
5. *reload* – reloads Open OpenGL renderers with new 3D models and updates Vuforia tracker mechanism with new trackable objects to finally prepare CARPs;
6. *select <category>* – selects semantic category provided by *AR Service Catalog*, where <category> stands for a concrete name of a category.

It is also worth to emphasize that *BrowsAR* supports Android internationalization functionality by recognizing voice commands stated in other languages. In the current implementation, the application adapts to the English and Polish languages – but it is easy to add an extension supporting other languages without modifying the source code. For instance, the application distinguishes "show map" and "pokaż mapę" commands that are used to show the map component. Every text string (and its different language equivalents) are listed in an Android string.xml file. The consequence of this approach is that there is no need to change the source code of the browser while adding new words supporting others languages. Due to the fact that Google's speech recognition service not always returns correct results, e.g., for saying 'map on' the result could be 'map own', voice commands have been declared as an XML string array [57], which includes frequent misinterpreted versions of the commands.

5.3. Server-side CARE components

5.3.1. *Semantic Augmented Reality Middleware*

Semantic Augmented Reality Middleware serves indirectly in support of *BrowsAR*, *CARE Modeler*, and external applications. It provides REST API that connects semantically described data sources to the client applications. The middleware fulfils two functions: (1) it manages knowledge about AR service providers, its resources, and contexts in which particular components of AR service providers can be composed to build CARPs dynamically; and (2) it creates and provides dCARP descriptions that meet criteria of the end-user context.

Figure 5.14 presents the three-layered architecture of *Semantic Augmented Reality Middleware*, in which each rounded rectangle presents a programming framework or library that was used to implement the software. *Semantic Augmented Reality Middleware* consists of the following three layers: *Network Layer*, *Data Layer*, and *Semantic Layer*. Each of these layers is described below.

Network Layer (NL) is responsible for communication with the clients. It provides five RESTful services as follows: *AR Service Register Point*, *Device Types Service*, *AR Service Catalog*, *Search Service*, and *Statements Service Endpoint*. To enable implementation of the above-mentioned services, Apache CXF and Spring frameworks have been used [14, 108]. CXF provides tools to develop RESTful services via annotations using the HTTP binding. Using URI templates and annotations, there is a possibility to bind a service operation to arbitrary URL and HTTP verb (such as GET, PUT, POST, and DELETE) combinations. For example, a getARServiceProviders method can be annotated with @Get and @HttpResource("/catalog/{category}"). CXF will then listen for GET requests on that URL and using the parameter at the *category* location as a parameter of the service will list all semantically described individuals of AR Service Providers. Moreover, CXF supports Spring XML syntax, making

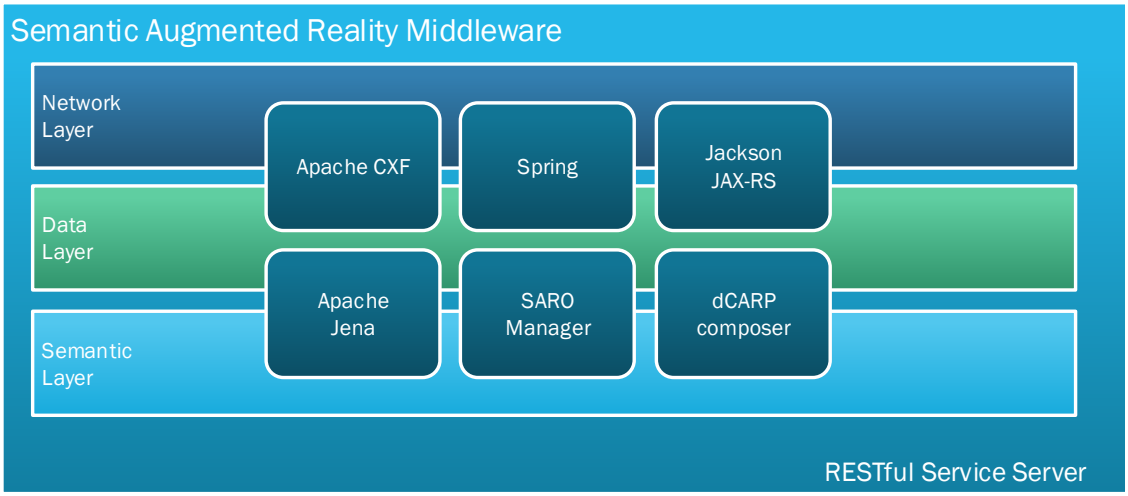


Figure 5.14: The architecture of *Semantic Augmented Reality Middleware*.

easy to declare endpoints which are backed by Spring.

Data Layer (DL) is responsible for two kinds of operation. First, it handles data received from the clients (through the Network Layer) by processing JSON content into Plain Old Java Objects (POJOs) using the Jackson JAX-RS library [7]. Second, it processes dCARP descriptions retrieved from the Semantic Layer by decoding them into JSON objects to send to the clients via the Network Layer.

Semantic Layer (SL) is responsible for extracting and writing data provided by the clients to the CARE knowledge base. To perform these operations, Apache Jena framework was chosen [15]. Furthermore, to retrieve semantic-based data, the SDMM method uses SPARQL queries that are executed against CARE knowledge base (an example is presented in Listing 5.7). The results of the SDMM method are packed into dCARP descriptions that are further handled by the Data Layer and finally are dispatched to the clients. Last but not least, SARO Manager is responsible for persisting semantic triples describing the CARE environment within the CARE knowledge base.

Listing 5.7: An example of SPARQL query performed within the Semantic Layer.

```
1  PREFIX owl: <http://www.w3.org/2002/07/owl#>
2  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
3  PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
4  PREFIX geo: <http://www.georss.org/georss/>
5  PREFIX gml: <http://www.opengis.net/gml/>
6  PREFIX saro: <http://www.semantic3d.org/dr/SARO#>
7  PREFIX daro: <http://www.semantic3d.org/dr/daro#>
8  PREFIX beacon: <http://www.semantic3d.org/dr/beacon#>
9  PREFIX dt: <http://www.semantic3d.org/dr/deviceType#>
10 PREFIX time: <http://www.w3.org/2006/time#>
11
12
13 SELECT ?arresource ?beacon ?devType ?temporalEntity ?start ?end
14   WHERE {
15     ?arresource saro:isIdentifiedBy ?beacon .
16     ?beacon beacon:uuid  "b9407f30-f5f8-466e-aff9-25556b57fe6d"^^xsd:normalizedString .
```

```
17      ?beacon beacon:major "19464"^^xsd:int .
18      ?beacon beacon:minor "30157"^^xsd:int .
19      ?arresource dt:isPresentableOn <http://www.semantic3d.org/dr/deviceType#Smartphone> .
20      <http://www.semantic3d.org/dr/daro#University> daro:encompasses ?arresource .
21      ?arresource saro:hasTemporalEntity ?temporalEntity .
22      ?temporalEntity time:hasBeginning ?hasBeginning .
23      ?temporalEntity time:hasEnd ?hasEnd .
24      ?hasBeginning time:inXSDDateTime ?start .
25      ?hasEnd time:inXSDDateTime ?end .
26
27      FILTER (?start <= "2017-03-22T10:00:39.254+01:00"^^xsd:dateTime &&
28                        ?end >= "2017-03-22T10:00:39.254+01:00"^^xsd:dateTime)
29    }
```

5.3.2. AR service providers

AR service providers are repositories of augmented reality resources that are used to compose diverse CARPs depending on the end-user's context. These distributed applications are characterized by independence from other elements of the CARE system. Their primary task is to deliver IRIs to AR resources. Moreover, AR service providers provide REST API that is used for persisting AR resources dispatched by *CARE Modeler*.

Figure 5.15 presents the two-layered architecture of AR service providers. Each AR service provider consists of: the Network Layer and the Data Layer. The technical implementation is very similar to the implementation of *Semantic Augmented Reality Middleware*. AR service providers also use Apache CXF and Spring as basic frameworks to implement RESTful services, as well as Jackson JAX-RS library to process JSON objects. However, AR service providers do not require the semantic layer since semantic data are handled by *Semantic Augmented Reality Middleware*. Particular layers of AR service providers are described below.

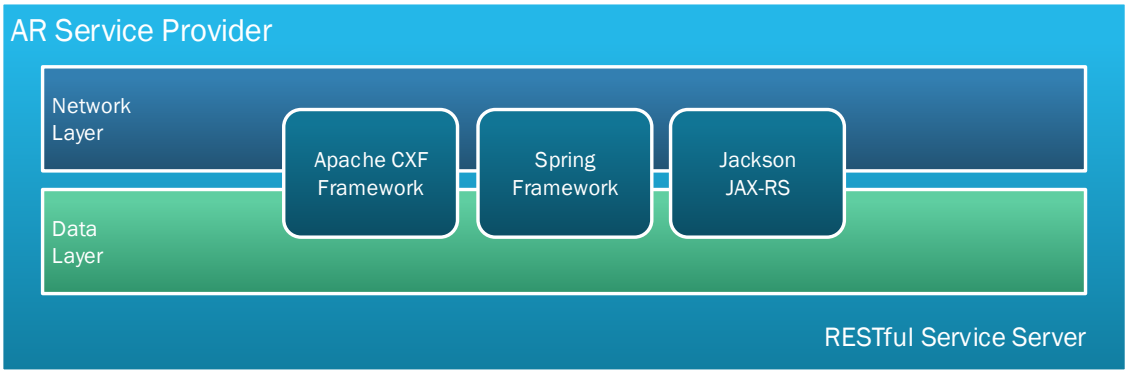


Figure 5.15: The architecture of AR service providers.

Network Layer (NL) is responsible for communication with *CARE Modeler* as well as *BrowsAR*. It provides three RESTful services responsible for persisting AR resources. Each AR resource is sent by a CARE environment designer via the *CARE Modeler* interface. Every multimedia content, sent via *CARE Modeler*, is encoded with the base64 algorithm. Further, these base64 representations are handled by the Data Layer. After decoding the representations and persisting multimedia content, trackables, content

objects, and data objects are assigned publicly available IRIs. Further, these IRIs are semantically described within *Semantic Augmented Reality Middleware*, and – in the end – are used as AR resources to dynamically compose CARPs in *BrowsAR*.

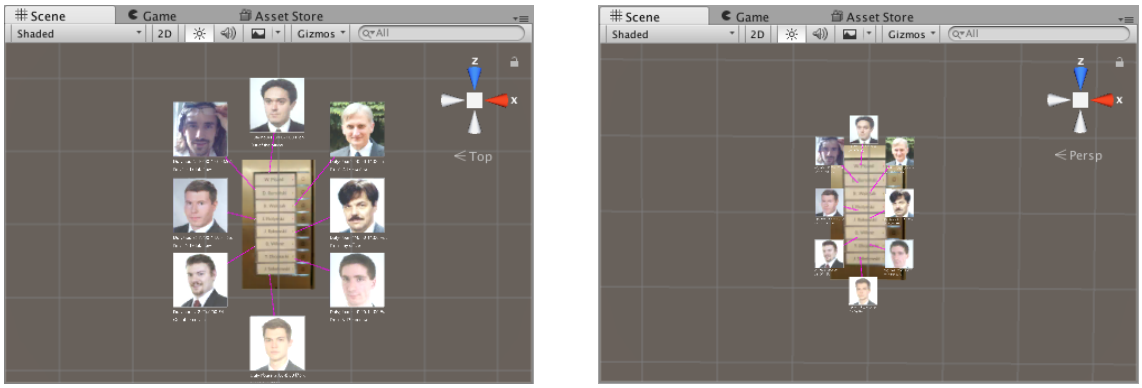
Data Layer (DL) is responsible in four types of operations. First, it processes data received from *CARE Modeler*, through the Network Layer, with the use of the Jackson JAX-RS library [7]. Second, it decodes JSON base64 contents into binary data. Third, it persists binary data by creating file output streams to write to the files. Finally, it constructs publicly available Internationalized Resource Identifiers (IRIs) for AR resources.

5.4. Examples

This section presents two use cases presenting contextual AR presentations modeled with the use of CARE.

5.4.1. AR Staff Members Information Service

The first example – 'AR Staff Information Service' – has been designed to provide useful information about university staff members. The service has been developed to augment the view of intercom devices with extra visual information about staff members – in this case university lecturers. The application displays lecturers' photos on a view of the intercom, where normally only lecturers' surnames are printed. Virtual photo-labels are linked with the real printed surnames. With this solution, end-users, e.g., students, can quickly associate a lecturer's face with the surname in order to make sure that they reach the person that they are looking for. The application renders also some extra information below a particular lecturer image, e.g., duty hours of the lecturer.



(a) Scene for smartphone.

(b) Scene for AR glasses.

Figure 5.16: Modeling two AR scenes with different transform properties.

Figure 5.16 shows two scenes presenting the same information, but displayed in different forms. The first scene is modeled for a smartphone device and the second for AR glasses. Due to the fact that the AR glasses devices have low-resolution cameras and the field of view is relatively small, graphical objects are modeled in a more compact way (Fig. 5.16b) comparing to the scene presented in Fig. 5.16a. Elements of the AR presentations were exported to an external AR service provider with the use of the CARE Modeler. Additionally, a particular beacon information was set in the user's context options. The use of beacons for context identification greatly enhances the scalability of the solution. Dozens of doors

(a) Smartphone view. (b) AR Glasses view. (c) A user experiences the AR presentation using AR Glasses.

Figure 5.17: AR Staff Members Information Service.

with labels can be unambiguously described, even if visually their representations differ only slightly. When a user approaches a particular context zone, the application downloads an appropriate set of visual markers to track, without having to identify them from the whole database with only visual properties. Also, the appropriate set of visual objects is downloaded based on the properties of the device class. Figure 5.17 depicts screenshots showing AR presentations displayed on a smartphone (5.17a) and AR glasses (5.17b).

5.4.2. AR City Events Information Service

The second example has been specially designed to inform and register a user in a novel way for a well-known event called *Researchers' night* taking place in many academic cities in Poland. Figure 5.18 presents the CARP consisting of four independent elements coming from AR service providers.

A fragment of the virtual map, a pin point, a button, and a trackable representing a poster have been combined to build an interactive AR presentation, whereby an end-user not only can experience information about the event, but can also register for participation with the use of an independent external service. The AR presentation has been modeled for a smartphone (Fig. 5.18a) and AR glasses (Fig. 5.18b and c), similarly as presented in the previous example. Additionally, Fig. 5.18c depicts a stereoscopic view taken from AR glasses.

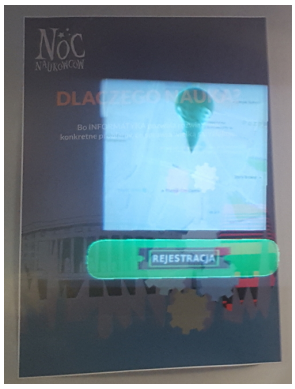
5.5. Summary

In this chapter the CARE system implementation has been described, explaining the client-side architecture and the main elements of the system. In order to build semantic modeling system for contextual augmented reality applications, a distributed software infrastructure has been designed and implemented. Working prototype of the CARE system proves the feasibility of the approach presented in Chapter 4.

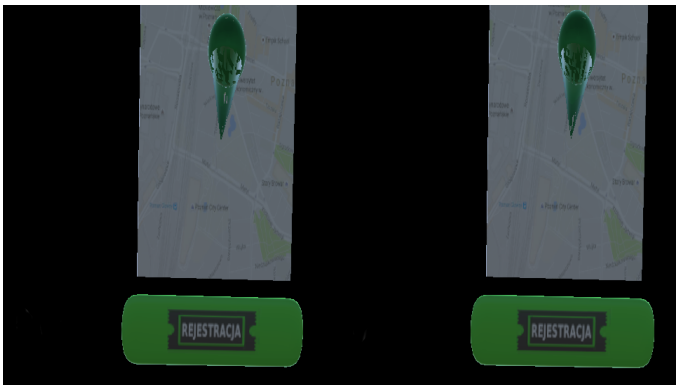
The presented architecture includes the main software elements such as *BrowsAR*, *CARE Modeler*, *Semantic Augmented Reality Middleware*, and multiple instances of AR service providers. The implementation is mainly based on the Java language (excluding *CARE Modeler* that is written in the C# language as an extension to the Unity3D game engine).

The CARE system is designed for the development of mobile AR experiences. The examples described in this chapter demonstrate usefulness of the CARE system for real-world applications.

(a) Smartphone.



(b) AR Glasses.



(c) A user experiences the AR presentation using AR Glasses.

Figure 5.18: AR City Events Information Service.

6. Evaluation of CARE

The chapter presents results of qualitative and quantitative evaluation of the CARE approach. A user study was conducted to test usefulness and easiness of use of CARE tools. Also, load testing was performed to investigate the CARE semantic search performance.

6.1. Qualitative evaluation

This section presents a qualitative user study that has been performed to evaluate usefulness and easiness to use of the *CARE Modeler* and *BrowsAR* applications while modeling and exploring contextual augmented reality environments. The following subsections cover design of the study, characteristics of participants that took part in the study, the collected results, and discussion.

6.1.1. Design of the study

The presented study consists of 3 parts that are listed below:

I. Learning and adopting new skills in the field of modeling AR presentations; deploying AR applications to mobile devices using Unity3D and Vuforia extension plugin; familiarizing with the CARE approach, *CARE Modeler* and *BrowsAR*;
II. Performing tasks related to modeling and exploring contextual AR environments;
III. Filling online questionnaire.

Part I: Learning and adopting new skills

First, the participants needed to learn how to develop a basic AR application using the Vuforia extension plugin to Unity3D. A tutor demonstrated how to model an AR presentation and how to deploy it on an Android device. At the same time, participants repeated activities and were asked to deploy their works as Android applications and test these applications on mobile devices. The goal of this assignment was to familiarize the participants with the Unity3D IDE and some essential features of Vuforia, such as image databases, image target, and AR camera, how to deploy an AR application, and how to test it.

After the participants acquired practical skills, the tutor introduced problems related to developing contextual AR environments and presented the CARE approach: what is the context; how it can be specified in CARE; how context can be used to describe resources, what is an AR service provider. Moreover, the tutor introduced beacon devices and explained how they can be used to describe contextual resources in case of indoor environments.

After the theoretical presentation, the participants were asked to download the *CARE Modeler* plugin and install it in Unity3D IDE. After the installation process, the tutor presented main assets of the tool, i.e., scripts, 3D models, and image databases, that can be used while modeling AR presentations, and explained particular parts of the graphical user interface of *CARE Modeler*. Next, the tutor developed an example context, which was used for describing resources of a created AR presentation and deployed

these elements to external AR service providers. Finally, the tutor demonstrated just-created contextual AR presentation within the *BrowsAR* mobile application. After the presentation, the participants were asked to develop a unique context and deploy their own contextual AR presentations using the *CARE Modeler* tool. The goal of this task was to familiarize the participants with the GUI and primary functions of the plugin.

Next task consisted in exploration of just-created AR environment by the participants. The tutor provided five smartphones with the *BrowsAR* application installed. The participants were asked to find and experience AR presentations that were created by them within the contextual AR environment. The goal of this task was to familiarize the participants with the AR interface and to demonstrate how easy is to locate and experience various contextual AR presentations created by the participants.

Part II: Performing tasks – standard tools vs. CARE tools

When the participants acquired the required skills, they were prepared to carry out tasks themselves. The participants were asked to model new AR presentations consisting of five 3D models derived from the assets of *CARE Modeler*, and to deploy it as an Android application to a mobile device using standard tools. The participants were asked to measure time from the beginning of building an Android application to the first experience of the AR presentation on a mobile device.

After the above-mentioned activity, the participants were asked to describe unique contexts in which AR resources of previously created AR presentation could be available to end-users to form CARPs. They were also asked to deploy contextually-described resources to external services and view their work in *BrowsAR*. Similarly, the participants were asked to measure the time from the beginning of deploying of a contextual AR presentation to the first experience of the created work within *BrowsAR*. Figure 6.1 depicts an example of an AR scene modeled by one of the participants.

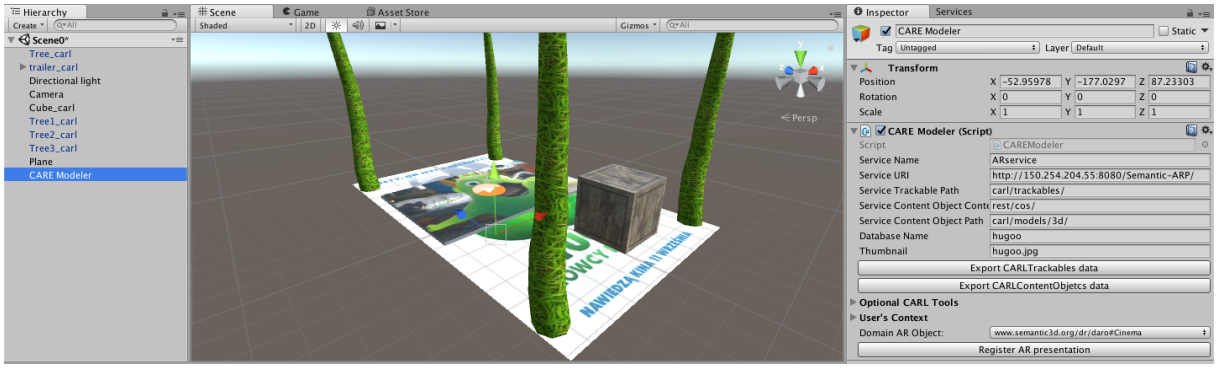


Figure 6.1: An example of an AR scene modeled by the one of the participants.

Part III: Filling online questionnaire

Finally, the participants were asked to fill-in an online questionnaire with the Likert scale rating items on a scale of 1 (uselessness/totally difficult/unintelligible) to 5 (critical/totally easy/totally intelligible).

6.1.2. Participants

All the participants took part in a facultative seminar whose topic was "Learn to design contextual augmented reality environment". A total of 14 test subjects (2 female and 12 male) participated in the study. However, four of them had to leave the class before the exploration task has begun. The subjects ranged in age from 19 to 22 years (M = 20.8 years, SD = 2.82 years).

Only two of them had some experience with the use of the Unity3D IDE.

6.1.3. Environment setup

In order to perform the user study, *CARE Modeler* was installed within Unity3D IDE 2017 by the participants. The software run within a virtual machine with Windows 8.1 Enterprise x64. The virtual machine consisted of two Intel Xeon E5-2660 v3 CPU 2.60GHz processors. The participants used Samsung Galaxy S6 devices with installed Android v. 7.0.

6.1.4. Results and analysis

Within this subsection, the results of the study are reported. They include, task completion times, questionnaire data gathered from the participants, as well as an analysis of the data, and discussion.

Task completion time

The time taken from deployment to first experience of an AR presentation using standard vs. CARE tools, as presented in Fig. 6.2, was analyzed using an independent t-test. The two-sample unequal variance (heteroscedastic) test was performed. The two-tailed distribution was used. The test confirmed that significant differences in the task completion times existed between using standard and CARE tools: T=6.69, p=0.0024 ($p < 0.05$).

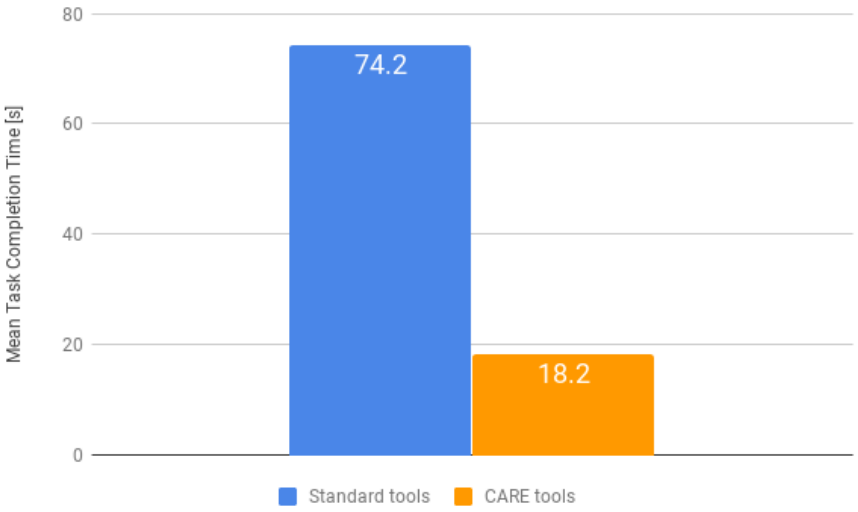


Figure 6.2: The mean completion time for completing the task with standard tools vs. CARE tools.

The time taken to complete the task was significantly shorter while using the *CARE Modeler* tool. Since Unity3D automatically starts an application after the process of building it, there was no time gap, e.g., to find an application to start it from the main menu of the Android system. The most time-consuming part, in the case of using standard tools, was the building process – statistically, it took almost 65% of the time. In the case of using CARE tools, the most time-consuming was the process of exporting AR resources (3D models, textures, and trackables) and the process of downloading these objects from AR service providers. What is important to note is that there was no need to recompile and rebuild *BrowsAR*. Theoretically, even if, the time of building an application would be removed from the calculation, it is still faster to experience an AR presentation using the CARE approach.

Questionnaire: Likert scale rating

After finishing the tasks related to Part II, the participants were asked to answer the questionnaire that included 9 questions (Tab. 6.1). The Cronbach's alpha indicated acceptable internal consistency among Likert items (α=.739; $0.8 > \alpha \geq 0.7$).

Q#	Evaluation of	M $\pm$ SD
Q1	*CARE Modeler* usefulness	4.2 $\pm$0.63
Q2	Easiness of installation	3.8 $\pm$1.03
Q3	Easiness of context description	3.7 $\pm$0.94
Q4	Easiness of trackables exportation	3.9 $\pm$0.73
Q5	Easiness of content objects exportation	4.0 $\pm$1.05
Q6	Easiness of AR resources registration	3.9 $\pm$0.74
Q7	*BrowsAR* usefulness	4.1 $\pm$0.74
Q8	Intelligibility of CARP modeling	4.0 $\pm$0.47
Q9	Practical applications of CARP	4.5 $\pm$0.52

Table 6.1: Likert scale rating questionnaire with means and standard deviation (scale 1-5).

The first six questions concerned the *CARE Modeler* usefulness as well as easiness of use of particular parts of the tool. To the first question Q1: "*Please assess the usefulness of CARE Modeler in modeling contextual AR environments.*" most of the participants responded that *CARE Modeler* was a very easy-to-use tool, which can be helpful for modeling contextual AR presentations. Three of them assessed that *CARE Modeler* was critical. The results of Q1 are presented in Figure 6.3a.

Concerning Q2: "*Please assess easiness of the installation process of CARE Modeler.*" two participants had a problem with the installation of the tool and assessed installation process as difficult. The rest assessed the process as easy/very easy.

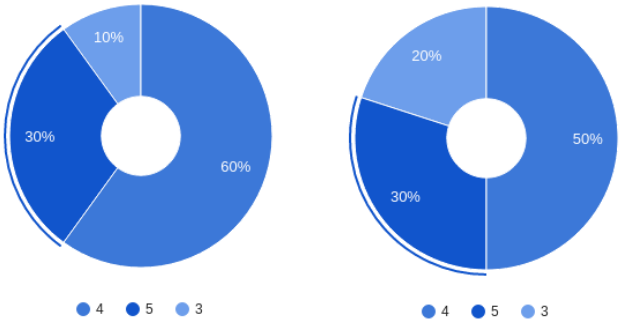
To the third question Q3: "*Please assess easiness of context description.*" two subjects responded that it was difficult to describe the context. This assessment could be an effect of entering beacon's identifiers in a wrong way. Each identifier consists of three components. By mistakenly entering an additional character or changing the number, the AR presentation elements are not correctly identified by a beacon. As a result, these participants had a problem with experiencing AR presentations created by them. For the rest, the process of describing the indoor context was very easy.

Answers to questions Q4: "*Please assess easiness of exporting content objects.*"; Q5: "*Please assess easiness of exporting trackables.*"; Q6: "*Please assess easiness of registration of AR resources.*" were assessed similarly – as very easy. In order to perform these tasks, the participants only needed to press the appropriate button and wait for the response results that appear as GUI messages.

The *BrowsAR* usefulness (Q7: "*Please assess the usefulness of BrowsAR.*") was assessed as very useful in most cases. Three subjects responded that the mobile application was a critical tool to explore the AR environment. The results of the Q7 are presented in Figure 6.3b.

Question Q8: "*Please assess intelligibility of the modeling contextual AR presentations.*" concerned whether the process of modeling contextual AR presentations was intelligible for the participants. The subjects assess the process as intelligible/very intelligible.

Finally, answer to question Q9: "*Do you think that the use of contextual AR applications can have a practical use?*" – was rated very high. The participants agreed that contextual AR presentations could have the practical use.



(a) *CARE Modeler* (b) *BrowsAR*

Figure 6.3: Aggregated ratings of evaluated CARE tools using Likert scale (1-useless, 2-slightly useful, 3-useful, 4-very useful, 5-critical).

6.1.5. Discussion

The obtained results are promising and demonstrate that non-experienced participants were able to finish tasks related to deployment and experience AR presentations in a shorter time using CARE tools in comparison with the use of standard tools provided by Unity3D. The tasks were performed without significant difficulties.

The collected data suggest that the *CARE Modeler* and the *BrowsAR* application were very useful in prototyping and experiencing contextual AR presentations. For three of them, the tools were rated as critical. None of the participants assessed usefulness level below three (useful). The results from questionnaire show that describing indoor context was either easy or very easy – only two subjects had a problem because they entered incorrect data and – as a result – they had a problem with location of their AR presentations while exploring the AR environment.

The following limitations of the presented quantitative user study must be considered. First, the research was performed within laboratory environment in which all CARE components worked together in intranet network with no internet latency issues. In a real-world case studies, the success of experiencing contextual AR presentations depends – on the one hand, on efficiency of searching and delivering dCARPs and, on the other hand, on the speed of providing AR resources by distributed AR service providers – both must be within an acceptable time. These components including the client-side – in most cases – will not work in the same local network, and in a case when an AR service provider does not perform very well, the overall time of building a particular CARP may be affected – including user experience. Due to distributed nature of the CARE environment, however, other CARPs will not be affected. Another issue is the limited sample size that can impact the confidence level of the study [45].

Moreover, the conducted user study has shown that non-experienced participants could acquaint themselves with the problem of building contextual AR environments and were able to learn how to model and experience contextual AR presentations – in most cases without significant difficulties. In addition, *BrowsAR* allowed participants not only to experience single self-created CARPs, but after switching the context, it was possible for them to access other CARPs created by other participants. In contrast to the standard Unity design process, the participants were often updating their AR presentations, and the results of the change were almost immediately visible in *BrowsAR*. What is important, the participants did not have to recompile the code – they were only focused on modeling and exploring AR presentations. It is worth to mention that the exploration activity has caused a lot of enjoyment for the participants.

6.2. Quantitative evaluation

This section presents a quantitative evaluation of the *Semantic Augmented Reality Middleware* performance. Load testing was performed under various conditions to determine *Semantic Augmented Reality Middleware* behavior – in particular the *Search Service* – that was configured in two different ways. Moreover, the completion time of the SDMM method was measured to compare the method's performance versus *Search Service* response time.

The following subsections cover design of the study, characteristics of the test plan, procedures and environment setup, followed by presentation of the performance results. Finally, discussion section is provided.

6.2.1. Design of the study

The purpose of this study was to test the response time performance of *Search Service* and SDMM with the use of a test plan imitating real-world user actions in large-scale CARE environments. The test plan takes into account various configurations including the number of users performing at the same time, ramp-up period, and duration of each test.

Table 6.2: Characteristics of the generated CARE knowledge bases.

KB	contexts	triples	size
1	10	1237	168K
2	100	3071	743K
3	1K	18371	6.2M
4	10K	171091	61M
5	100K	1701371	604M

In order to perform the load testing, five test CARE knowledge bases were generated differing in the number of context zones and the overall size of the triple base. Table 6.2 presents characteristics of the created CARE KBs that have been provided to the *Semantic Augmented Reality Middleware* while testing its performance. Each CARE knowledge base was built on the basis of contextually described AR resources that have been pseudo-randomly generated. In turn, each AR resource was associated with a pseudo-randomly generated context including a geographical location contained within the following ranges:

- latitude: [52.378509, 52.440262];
- longitude: [16.845494, 16.999999].

Figure 6.4 visualizes geographical boundaries of the above ranges.

6.2.2. Test plan

Firstly, a parametrized test plan was designed with the use of JMeter [16]. The test plan describes a series of steps imitating user actions executed against the middleware while exploring an AR environment. Due to the fact that *Search Service* is a core element responsible for searching semantically described AR resources on the basis of the user's context, the service was tested in various configurations. The test plan allows one to set:

- the number of users (threads),

Figure 6.4: Visualization of geographical boundaries for the created test CARE knowledge bases.

- the ramp-up period,
- the duration of the test.

Each simulated user process executed the test plan in its entirety and independently of other users. The ramp-up period expresses how long does it take to "ramp-up" to the full number of users chosen. For instance, if 40 users test is performed and the rump-up period is 120 seconds, then jMeter will take 120 seconds to run all 40 simulated user's processes. Thus, each process will start 3 seconds after the previous has been launched.

Listing 6.1: The template of HTTP message body representing a user's context.

```
1  {
2    "location":{
3      "latitude":52.${RANDOM_LATITUDE_VAR},
4      "longitude":16.${RANDOM_LONGITUDE_VAR}
5    },
6    "range":${RANDOM_RANGE_VAR},
7    "domain":${RANDOM_DOMAIN_VAR},
8    "deviceType":${RANDOM_DEVICE_TYPE_VAR},
9    "userDateTime":${RANDOM_DATE_VAR}
10 }
```

In order to test *Search Service* response times, a template of HTTP message body representing a simulated user's context was developed (as shown in Listing 6.1). The description consists of a geographical location (lines 2-5), range (line 6), domain (line 7), device type (line 8), and date/time (line 9).

The template uses randomized variables to generate user context. The following variables take pseudo-random values in the specified ranges:

- RANDOM_LATITUDE_VAR: 378509 to 440262,
- RANDOM_LONGITUDE_VAR: 845494 to 999999,
- RANDOM_RANGE_VAR: 100 to 600.

Additionally, values of RANDOM_DOMAIN_VAR and RANDOM_DEVICE_TYPE_VAR are randomly generated on the basis of subclasses of the *Domain* and the *DeviceType* classes from Δ and Θ ontologies. Lastly, the RANDOM_DATE_VAR variable takes pseudo-random date and time.

On the basis of the template, jMeter generates HTTP request messages imitating users' activity. Listing 6.2 presents an example HTTP request message having in its body generated user's context generated by jMeter.

Listing 6.2: An example of pseudo-random end-user context encoded with JSON.

```
POST http://www.semantic3d.org/Semantic-ARP/rest/catalog/search HTTP/1.1
Connection: keep-alive
Content-Type: application/json
Content-Length: 486
Host: 150.254.204.55:8080
User-Agent: Apache-HttpClient/4.5.3 (Java/1.8.0_73)
{
   "location":{
      "latitude":52.437491,
      "longitude":16.918441
   },
   "range":564,
   "domain":"http://www.semantic3d.org/dr/daro#Shop",
   "deviceType":"http://www.semantic3d.org/dr/deviceType#Smartphone",
   "userDateTime":"2018-01-10T10:30:10.2"
}
```

6.2.3. Procedures and environment setup

Two experiments were carried out using the developed test plan. Each experiment consisted of a series of tests that were configured in different ways.

The goal of the first experiment was to analyze the performance of *Search Service* and track which configuration meets satisfied and tolerated user satisfaction levels specified in [129], when using the CARE knowledge base with 1.701.371 triples (KB=5). Such a large-scale knowledge base can correspond to multi-domain CARE environment configured for a medium-size city. In this experiment, *Search Service* was tested in 11 configurations differing in the number of users performing at the same time. Each test lasted 60 seconds, while the ramp-up period was set to 0.5 seconds. The SDMM method has been set to the so-called *firstEnds* mode in which it returns only the first dCARP that is within the user's context – leaving aside the search for the best result. Within the experiment, the following statistical data were collected: the number of samples, mean, standard deviation, median, 90th, 95th, 99th percentile, minimum, maximum of the response time of *Search Service*, and the Apdex index. Finally, after running each test, an internal tool of the middleware was executed to collect the same type of data related only to the SDMM method to compare them with the data describing the performance of the whole *Search Service*.

The goal of the second experiment was to compare the performance of the *Search Service* in relation to the size of a particular CARE knowledge base. Within this experiment, CARE knowledge bases from 1 to 5 have been used (table 6.2). The *Search Service* was configured in two ways, i.e., by returning the first result – similarly as SDMM was set in the previous experiment (the *firstEnds* mode) – and by returning the optimal result (the *optimalEnds* mode). The configuration of the test plan was the same for each test, i.e., 100 users; 0.5 sec. for the ramp-up period; each test lasted 60 sec. After the completion of each test plan, statistical data were collected including: number of samples, mean, standard deviation

median, 90th, 95th, 99th percentile, minimum, maximum of the response time of *Search Service*, and the Apdex index.

The Apdex index was calculated using the following formula:

$$Apdex = \frac{C(s) + \frac{C(t)}{2} + 0 * C(f)}{C(a)}, \tag{6.1}$$

where:

- $C(s)$ – the number of samples of $s_1, s_2, ..., s_l$ in which response time is less or equal T time (the so-called $satisfied$ level);
- $C(t)$ – the number of samples of $t_1, t_2, ..., t_m$ in which response time is greater than T time and less or equal 4T (the so-called $tolerated$ level);
- $C(f)$ – the number of samples of $f_1, f_2, ..., f_n$ in which response time is greater than 4T time (the so-called $frustrated$ level);
- $C(a)$ – the sum of all samples $(C(s) + C(t) + C(f))$.

Taking into account specificity of mobile network applications, the $satisfied$ level has been set to 1 sec. (10^3 ms) and the $tolerated$ level to 10 sec. (10^4 ms), following Nielsen [98] research on the subject.

Semantic Augmented Reality Middleware has been installed on Apache Tomcat Server v.8.0.41 run within Ubuntu v.15.10 i686 with installed Java v.1.8.0_73. Java was set up with the following parameters: -Xms2048m -Xmx2048m. The computer was equipped with in an Intel Core i7-2600K CPU 3.40GHz eight-core processor.

6.2.4. Results and analysis

This subsection presents the collected results of quantitative evaluation as well as an analysis of two conducted experiments.

Experiment 1

Table 6.3 presents the collected results of Experiment 1. The unit of columns 4 to 11 is millisecond. The first test simulated 10 users, the second – 50, and each next test was increased by 50 users.

Table 6.3: Collected results of the Experiment 1 – *Search Service*.

Test	Users	Samples	Mean	SD	Median	90th	95th	99th	Min	Max	Apdex
1	10	10429	57	119	21	135	222	604	2	2608	0.999
2	50	11072	271	548	103	647	1080	2698	2	13614	0.972
3	100	10647	566	985	250	1371	2176	4852	2	17116	0.925
4	150	10640	852	1479	374	2095	3320	7341	2	36210	0.882
5	200	11415	1062	2044	440	2568	4159	9605	2	52698	0.858
6	250	10686	1424	2696	610	3358	5653	12467	2	62094	0.813
7	300	10830	1694	3477	668	3985	6830	16726	2	66746	0.797
8	350	11557	1852	3352	805	4520	7417	15300	2	62934	0.767
9	400	12054	2039	3720	887	4933	7958	18929	2	58119	0.751
10	450	12136	2289	4023	989	5604	9201	19875	2	65841	0.731
11	500	11991	2568	4367	1118	6386	10348	21784	2	66596	0.708

First of all, what can be noted in the presented data, is the increase of response time in relation to the number of users performing in each test. Consequently, the Apdex index decreases with each subsequent test. The *satisfied* level is exceeded by about 62 milliseconds of the mean response time for the test number 5. The rest of mean response times are within the *tolerated* level.

The standard deviation (SD), in most cases, is significantly greater than the mean response time signaling that SD (and also the mean) are affected by high results of response times in the data set. These outliers may result from long-lasting processes of matching the geographical location of a user to AR resources in specific cases.

To give a broader view of obtained results the 50^{th}, 90^{th}, 95^{th}, and 99^{th} percentiles of response time were analyzed. In each test, the median is significantly lower than the mean time. For 50% of samples the service execution took less than 10^3 ms for tests from 1 to 10. For more than 90% of samples the service execution took less than 10^4 ms for all tests. Only for 5% of samples of test number 11 the response time exceeded the *tolerated* level.

Table 6.4: Collected results in the Experiment 1 – *SDMM method*.

Test	Users	Samples	Mean	SD	Median	90th	95th	99th	Min	Max	Apdex
1	10	10429	56	117	20	133	221	577	1	2605	0.999
2	50	11072	270	548	102	646	1079	2701	1	13614	0.972
3	100	10647	565	985	248	1369	2175	4866	1	17115	0.925
4	150	10640	850	1479	372	2094	3320	7290	1	36205	0.882
5	200	11415	1059	2043	434	2566	4158	9611	1	52696	0.858
6	250	10686	1418	2696	604	3359	5653	12503	1	62050	0.814
7	300	10830	1685	3475	657	3977	6836	16733	1	66746	0.798
8	350	11557	1841	3352	792	4516	7422	15366	1	62932	0.769
9	400	12054	2029	3720	871	4917	7964	18939	2	58118	0.752
10	450	12136	2277	4023	971	5597	9202	19915	1	65693	0.733
11	500	11991	2560	4367	1109	6383	10347	21800	1	66596	0.71

In parallel, the same kind of data concerning the performance of the SDMM method (excluding other elements of the CARE middleware service), have been collected (as presented in Table 6.4). The results are convergent for all tests when comparing them to the data from Table 6.3 – the values are slightly lower. The difference may indicate a small latency associated with transmitting messages over the network.

Figure 6.5 presents the mean response time in relation to the number of users. What can be noticed is the near linear relationship between the mean response time of *Search Service* and the number of users.

In turn, Figure 6.6 presents the Apdex index in relation to the number of users. The diagram shows relatively small decrease of Apdex in relation to the number of users. However, the Apdex index is still within the *satisfied* level even for hundreds of concurrent users, suggesting that *Search Service* performed very well while conducting Experiment 1.

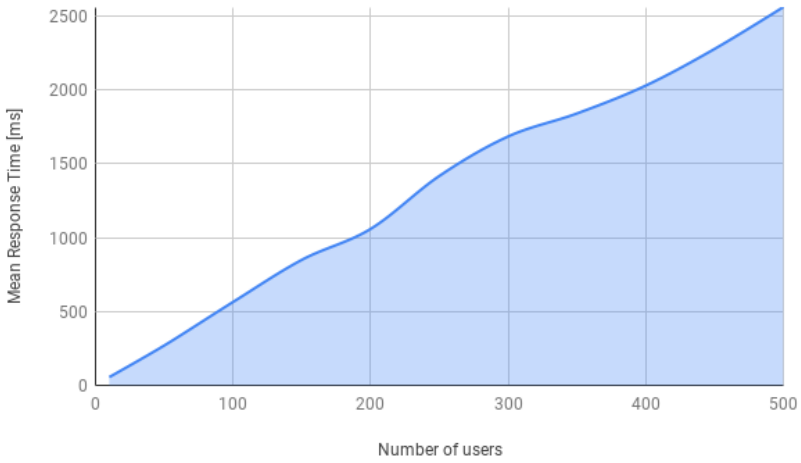


Figure 6.5: Relationship between mean response time and the number of users.

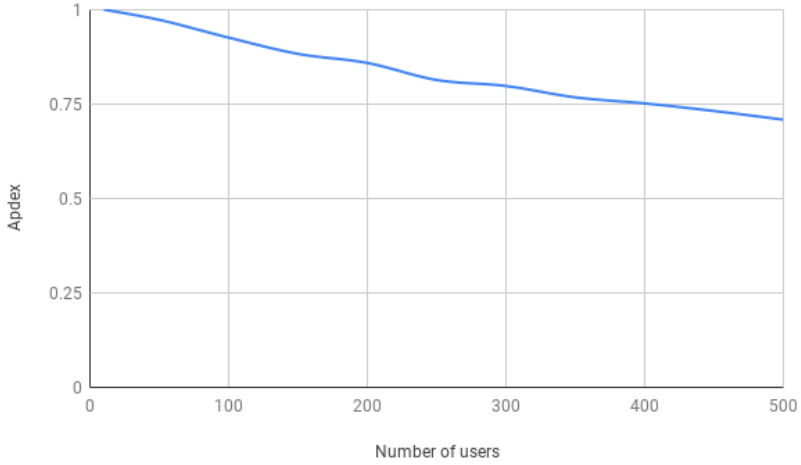


Figure 6.6: The Apdex index of the *Search Service*.

Experiment 2

Tables 6.5 and 6.6 present results obtained in Experiment 2 – the *Search Service* performance related to the use of different CARE knowledge bases configured by means of the *firstEnds* and *optimalEnds* modes, respectively.

What can be first noticed in both tables (6.5 and 6.6), is the rise of the mean response time in relation to the size of CARE knowledge bases. For the *firstEnds* mode, the increase is in an acceptable limits and almost all results stay under the *satisfied* level for each knowledge base. Only 1% of data of the test with KB=5 exceeded 4957ms (about 5 sec.).

In contrast, in the case of the *optimalEnds* mode, the test with KB=5 (Table 6.6) shows that most of the results exceeded the *tolerated* level. This may be due to the limited capabilities of the used hardware equipment (a regular PC). To verify the possibility of using very large knowledge bases (at the level of hundreds of thousands context ranges), the test has been repeated on enterprise-level server infrastructure.

Table 6.7 presents the results obtained from the repeated Experiment 2 performed on a virtual machine installed on a cluster with 4 Intel Xeon E7-2830K CPU 2.13GHz four-core processors running

on Windows Server 2008 x64 with installed Java v.1.8.0_73. The machine was equipped with 32 GB of RAM. In that case, the performance of *Search Service* configured in the *optimalEnds* mode was significantly better when compared with data from Table 6.6 – especially for the largest knowledge base (KB=5) with 10^5 context ranges. At least 50% of results are within the *tolerated* level.

Table 6.5: Results of *Search Service* configured in the *firstEnds* mode on PC.

KB	Samples	Mean	SD	Median	90th	95th	99th	Min	Max	Apdex
1	262411	9	27	1	22	60	140	0	449	1.000
2	163246	34	63	2	117	167	284	0	773	1.000
3	41164	144	134	111	328	405	569	1	1288	1.000
4	13495	443	560	229	1258	1776	2421	1	4488	0.933
5	11131	539	1034	224	1286	2177	4957	1	20457	0.930

Table 6.6: Results of *Search Service* configured in the *optimalEnds* mode on PC.

KB	Samples	Mean	SD	Median	90th	95th	99th	Min	Max	Apdex
1	232918	11	30	1	28	69	158	0	454	1.000
2	154974	36	66	2	121	176	295	0	846	1.000
3	29970	199	140	176	389	463	616	2	1092	1.000
4	3152	1912	484	1877	2537	2758	3266	610	3883	0.508
5	389	18917	1930	18695	20437	21499	28592	14509	33270	0.000

Table 6.7: Results of *Search Service* configured in the *optimalEnds* mode on enetrprise server.

KB	Samples	Mean	SD	Median	90th	95th	99th	Min	Max	Apdex
1	179699	2	5	0	15	16	16	0	110	1.000
2	145404	3	6	0	16	16	16	0	172	1.000
3	54117	98	230	15	405	608	1029	0	3183	1.000
4	6162	968	310	967	1279	1404	1992	31	4306	0.793
5	698	9136	972	9220	10311	10608	11218	5319	11856	0.413

Figure 6.7 presents results obtained on two different computer systems. What can be observed, is the near linear relationship between the mean response time and the number of context ranges – especially for knowledge bases with 1000 context ranges and larger. This observation means that the CARE system is scalable and can be used for very large knowledge bases, provided that appropriate hardware and software environment is employed.

6.2.5. Discussion

The findings suggest that an application of semantic web techniques can be an efficient solution to search contextually described distributed resources constituting interactive AR presentations. Two performed experiments have shown that semantically described AR resources can be selected, matched, and provided in an acceptable time. The *Search Service* was able to perform well, even for large

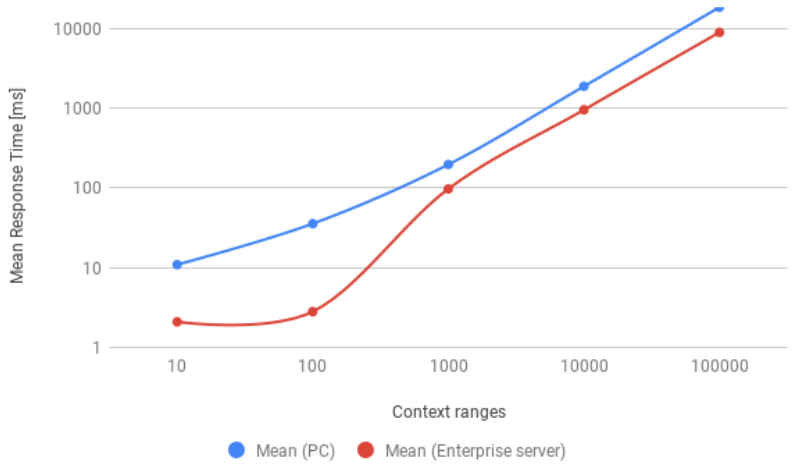


Figure 6.7: Relationship between the mean response time and the number of context ranges using the *optimalEnds* mode for PC and enterprise server.

knowledge bases – up to several hundreds thousands context ranges – which can cover a medium-size city with AR presentations in multiple domains.

Increasing the number of users performing at the same time caused a slight decrease of performance of *Search Service*, as well as SDMM in Experiment 1. In the second experiment, the size of a particular knowledge base had an impact on the final results. The mean response time near linearly increased in relation to the number of users and the number of context ranges, indicating that the CARE system is scalable and can be used for very large knowledge bases.

What is important in the presented approach, is that separate knowledge bases can be created for different application domains and different geographical regions, thus limiting the maximum necessary size of the KB and greatly improving performance of the *Search Service*. Also, the SARO ontology could be extended with a City ontology providing classes, such as District, Suburbs, Streets, Address, etc., by means of which a more precise description of context could be possible.

The presented results contribute to filling the gap concerning the performance evaluation of augmented reality systems supported with semantic web techniques. While the solutions introduced within Chapter 2 Section 3.3.2 look promising, little, or no attention has been paid in the existing literature regarding to the quantitative analysis of the performance of presented works.

7. Conclusions

Despite the current success of the augmented reality technology, including significant progress in the development of AR applications and the rapid growth of the AR market, many research areas related to AR have not yet been sufficiently addressed. One of such areas is the dynamic contextual composition of AR presentations based on data coming from independent distributed sources. The CARE approach, which is the main contribution of this dissertation, goes beyond the current state of the art in the field of AR by providing a generic solution to the problem of building ubiquitous contextual AR environments, in which AR presentations are not dependent on a specific application, platform or device, but elements forming AR experiences are independently provided by the available external sources. In CARE, the key mechanism responsible for automatic selection and composition of AR resources is based on the user's context. The context reflects which real-world objects should be augmented and with what kind of content, taking into user's preferences, date, time, outdoor and indoor location, the user's device type and its capabilities.

The presented solution detaches AR applications from specific multimedia content, data, and trackables, which now may be provided by independent AR service providers. Moving AR resources outside applications permits to overcome the main limitation of current AR applications, i.e., fragmentation of functionality between various specific applications. With the use of CARE, users can freely experience AR presentations in a continuous and contextual manner using a single browser.

The possibility of combining in a single AR presentation resources coming from different distributed service providers enables building a new class of ubiquitous AR systems and is a key to application AR technology to new application domains. For example, in a city AR service, trackable markers may come from municipal services, content objects from different providers of AR services, while scenarios from application developers. Only in such a heterogeneous environment, practical ubiquitous AR environments may be realized.

The "heart" of CARE – the SARO ontology – is based on a generic data model that enables to link AR resources and service providers to various elements of the user's context. SARO is built on top of the well-known standards, such as the Geography Markup Language Encoding Standard (GML) and W3C OWL-time ontology. The use of standards facilities adoption of the presented ontology in other systems in which the user's context plays an important role. In addition, in the context description, SARO not only supports outdoor positioning, but also indoor environments by the use of Indoor Position Detection Ontology. CARE enables to semantically describe AR resources and service providers that can be further searched, matched, combined, and presented within rich context-dependent AR presentations. This functionality is provided by the SDMM method. Moreover, SDMM allows users to re-search, re-match, re-combine, and re-present AR presentations based on the continuously changing user's context.

CARE provides easy-to-use software components: *CARE Modeler*, *BrowsAR*, *Semantic Augmented Reality Middleware*, and AR service providers supported with semantics, which can be used in modeling as well as exploring large-scale CARE environments – by designers that do not require to have technical skills and end users accessing the presentations.

The obtained evaluation results, presented in Chapter 6, contribute to filling the gap concerning the performance evaluation of augmented reality systems supported with semantic web techniques. These findings suggest that application of semantic web techniques is a viable solution to the problem of building dynamic contextual AR presentations. Two performed quantitative experiments have shown that semantically described data can be selected, matched, and provided in a reasonable time – not disturbing end users.

This dissertation makes several major contributions to the field of augmented reality. These contributions are as follows:

1. A literature review of augmented reality technology including application of semantic web techniques to AR. The focus of this review is on the diversity of AR application domains including areas supported with semantics. Also, a review of AR software packages and frameworks widely used for building AR applications has been provided.
2. The CARE Architecture, which enables execution of complex and extensive computation of semantic processing of distributed AR resources on the server-side, while contextual AR presentations are designed, dynamically formed and rendered in real time on the client-side.
3. Formal definition of *Contextual Augmented Reality Environment*. The model formalizes the concepts of various types of AR resources, AR service providers, CARE user's context, contextual AR presentation, its description, and the CARE environment.
4. *Semantic Augmented Reality Ontology* used to model large-scale contextual distributed augmented reality environments. SARO covers various elements of the user context, such as preferences, time, date, indoor and outdoor locations, a device type, its capabilities, and also context-depended AR resources and AR service providers.
5. A novel declarative language, called *Contextual Augmented Reality Language*, that supports modeling contextual AR environments.
6. *Semantic Discovery and Matching Method* – the method and algorithm of semantic discovery and matching AR resources constituting contextual AR presentations.
7. Software components: *CARE Modeler*, *BrowsAR*, *Semantic Augmented Reality Middleware*, and multiple AR service providers.
8. The qualitative user study demonstrating that non-experienced users are able to perform the tasks of AR environment modeling and exploring without significant difficulties in a shorter time with the use of the proposed software tools in comparison to standard tools. The collected data suggest that *CARE Modeler* and *BrowsAR* were useful for prototyping and experiencing contextual AR presentations.
9. The performance evaluation of the CARE system including – inter alia – the mean response time, throughput, the Apdex index – taking into account multiple test configurations differing in the number of simulated users performing at the same time as well as knowledge bases of different sizes.

To summarize, the goal of this dissertation has been fully achieved. Performed experiments and obtained results presented in Chapter 6 prove that *the CARE approach enables efficient modeling of large-scale contextual distributed augmented reality environments*.

There are areas of potential future research. First, the limitations discussed in Chapter 6 can be addressed. A qualitative user study is recommended to be repeated outside the laboratory environment, i.e., in such conditions in which all software components including elements of the server and the client sides work in different networks – to explore how internet latency may affect experience of users performing in a CARE environment. While the CARE system has succeeded in laboratory conditions, there is still a pending question how it would perform in an open real-world environment.

Second, since AR glasses are not yet available and accessible for mass usage, further research on the usability of CARE on such devices is needed. In the near future when this kind of devices will have a chance to be widely adopted in the daily use – just as mobile phones today – it will be important to conduct experiments how users perform in CARE with this type of devices.

Bibliography

[1] *Apple App Store*. https://itunes.apple.com, 2016.

[2] *AREL: Augmented Reality Experience Language*. https://my.metaio.com/dev/arel/index.html, 2015.

[3] *Aurasma*. http://www.aurasma.com, 2014.

[4] *Aurasma Studio*. https://studio.aurasma.com, 2015.

[5] *CoreAR for iOS*. https://github.com/sonsongithub/CoreAR, 2015.

[6] *Google Play*. https://play.google.com, 2016.

[7] . *JAX RS Provider For JSON Content Type*. https://mvnrepository.com/artifact/org.codehaus.jackson/jackson-jaxrs.

[8] *Layar Creator*. https://www.layar.com/creator/, 2015.

[9] *NyARToolkit for Java*. https://github.com/nyatla/NyARToolkit/, 2015.

[10] *Wikitude Developer*. http://www.wikitude.com/developer, 2015.

[11] *Wikitude Studio*. http://studio.wikitude.com/, 2015.

[12] Chris Aart, Bob Wielinga, and Willem Robert Hage. *Knowledge Engineering and Management by the Masses: 17th International Conference, EKAW 2010, Lisbon, Portugal, October 11-15, 2010. Proceedings*, chapter Mobile Cultural Heritage Guide: Location-Aware Semantic Search, pages 257–271. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[13] Mario Alberto Chavez Anduaga. Lux-ar, 2014.

[14] Apache Software Foundation. *Apache CXF Home Page*. http://cxf.apache.org/, 2017.

[15] Apache Software Foundation. *Apache Jena Semantic Web Framework*. http://jena.apache.org/, 2017.

[16] Apache Software Foundation. *JMeter*. http://jmeter.apache.org/, 2017.

[17] Apple Inc. *ARKit overview*. https://developer.apple.com/arkit/, 2017.

[18] ARToolworks. Artoolworks, 2015.

[19] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. Dbpedia: A nucleus for a web of open data. In *The semantic web*, pages 722–735. Springer Berlin Heidelberg, 2007.

[20] Autodesk. Fbx - adaptable file format for 3d animation software, 2016.

[21] Benjamin Avery, Wayne Piekarski, James Warren, and Bruce H. Thomas. Evaluation of user satisfaction and learnability for outdoor augmented reality gaming. In *Proceedings of the 7th Australasian User Interface Conference - Volume 50*, AUIC '06, pages 17–24, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc.

[22] Betül Aydin, Jérôme Gensel, Philippe Genoud, Sylvie Calabretto, and Bruno Tellez. An architecture for surroundings discovery by linking 3d models and lod cloud. In *Proceedings of the Second ACM SIGSPATIAL International Workshop on Mobile Geographic Information Systems*, MobiGIS '13, pages 9–16, New York, NY, USA, 2013. ACM.

[23] Ronald T Azuma et al. A survey of augmented reality. *Presence*, 6(4):355–385, 1997.

[24] Tim Berners-Lee, James Hendler, Ora Lassila, et al. The semantic web. *Scientific american*, 284(5):28–37, 2001.

[25] Mark Billinghurst. Shared space: Collaborative augmented reality. In *ACM SIGGRAPH 99 Conference Abstracts and Applications*, SIGGRAPH '99, pages 178–, New York, NY, USA, 1999. ACM.

[26] Christian Bizer, Tom Heath, Kingsley Idehen, and Tim Berners-Lee. Linked data on the web (ldow2008). In *Proceedings of the 17th International Conference on World Wide Web*, WWW '08, pages 1265–1266, New York, NY, USA, 2008. ACM.

[27] T. Blum, R. Stauder, E. Euler, and N. Navab. Superman-like x-ray vision: Towards brain-computer interfaces for medical augmented reality. In *Mixed and Augmented Reality (ISMAR), 2012 IEEE International Symposium on*, pages 271–272, Nov 2012.

[28] Tobias Blum, Valerie Kleeberger, Christoph Bichlmeier, and Nassir Navab. mirracle: An augmented reality magic mirror system for anatomy education. In *Virtual Reality Short Papers and Posters (VRW), 2012 IEEE*, pages 115–116. IEEE, 2012.

[29] Su Cai, Xu Wang, and Feng-Kuang Chiang. A case study of augmented reality simulation system application in a chemistry course. *Computers in Human Behavior*, 37:31–40, 2014.

[30] Jacques Cohen. *Communications of the ACM - Special issue on computer augmented environments: back to the real world*, 36(7), 1993.

[31] Total Immersion Company. *Augmented Reality Virtual Fitting Room*.

[32] Total Immersion Company. *TryLive Virtual Eyewear Try-on*.

[33] Open Geospatial Consortium and W3C. Time ontology in owl. On-line, 2017.

[34] Douglas Crockford. Javascript object notation (json), 2015.

[35] Mercedes Bentz R & D. *Head-Up Display for Mercedes Bentz*.

[36] DAQRI. Daqri, 2015.

[37] DBpedia community). Dbpedia sparql endpoint.

[38] Lucio Tommaso De Paolis and Giovanni Aloisio. Augmented reality in minimally invasive surgery. In SubhasChandra Mukhopadhyay and Aimé Lay-Ekuakille, editors, *Advances in Biomedical Sensing, Measurements, Instrumentation and Systems*, volume 55 of *Lecture Notes in Electrical Engineering*, pages 305–320. Springer Berlin Heidelberg, 2010.

[39] Chris Dede. Theoretical perspectives influencing the use of information technology in teaching and learning. In Joke Voogt and Gerald Knezek, editors, *International Handbook of Information Technology in Primary and Secondary Education*, volume 20 of *Springer International Handbook of Information Technology in Primary and Secondary Education*, pages 43–62. Springer US, 2008.

[40] Digi-Capital Blog. Ar and vr startups raise record 3.6 billion in last 12 months as market transition accelerates.

[41] Tobias Domhan. Android augmented reality - andar, 2013.

[42] J. Ehara and H. Saito. Texture overlay for virtual clothing based on pca of silhouettes. In *Mixed and Augmented Reality, 2006. ISMAR 2006. IEEE/ACM International Symposium on*, pages 139–142, Oct 2006.

[43] P. Eisert, P. Fechteler, and J. Rurainsky. 3-d tracking of shoes for virtual mirror applications. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–6, June 2008.

[44] Kudan AR Engine. Kudan ar engine, https://www.kudan.eu/, 2015.

[45] Jorge Faber and Lilian Martins Fonseca. How sample size influences research outcomes. *Dental press journal of orthodontics*, 19(4):27–29, 2014.

[46] M. Fiala. Magic mirror system with hand-held and wearable augmentations. In *Virtual Reality Conference, 2007. VR '07. IEEE*, pages 251–254, March 2007.

[47] Roy Fielding. *Representational State Transfer (REST)*. http://www.ics.uci.edu/˜fielding/pubs/dissertation/rest_arch_style.htm

[48] Apache Software Foundation. Apache license, version 2.0, 2004.

[49] Free Software Foundation. Gnu general public license, 2007.

[50] Elisabeth Freeman, Eric Freeman, Bert Bates, and Kathy Sierra. *Head First Design Patterns*. O' Reilly & Associates, Inc., 2004.

[51] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[52] DoPanic GmbH. Panicar, 2015.

[53] Daniele Gobbetti, Miriam Fostini, and Hannes Boran. Mixare – the open source augmented reality engine, 2013.

[54] Google Inc. *Android Studio - the offical IDE for Android*. https://developer.android.com/studio/index.html.

[55] Google Inc. *Gson – an open source Java library to serialize and deserialize Java objects to (and from) JSON*. https://github.com/google/gson.

[56] Google Inc. *Map Style Wizard*. https://mapstyle.withgoogle.com/.

[57] Google Inc. *String Array*. https://developer.android.com/guide/topics/resources/string-resource.html.
[58] Google Inc. *ARCore overview*. https://developers.google.com/ar/discover/, 2017.
[59] Google Inc. *Google Maps API for Android*, 2017.
[60] Khronos Group. Collada - 3d asset exchange schema, 2004.
[61] P. Grimmand M. Haller, V. Paelke, S. Reinhold, C. Reimann, and R. Zauner. Amire - authoring mixed reality. In *Proc. of the First IEEE International Augmented Reality Toolkit Workshop.*, page 2, 2002.
[62] Anne-Cecilie Haugstvedt and John Krogstie. Mobile augmented reality for cultural heritage: A technology acceptance study. In *Mixed and Augmented Reality (ISMAR), 2012 IEEE International Symposium on*, pages 247–255. IEEE, 2012.
[63] Simon Heinen. Droidar - a framework for augmented reality on android, 2015.
[64] Ramón Hervás, Alberto Garcia-Lillo, and José Bravo. *Ambient Assisted Living: Third Int. Workshop, IWAAL 2011, Held at IWANN 2011, Torremolinos-Málaga, Spain, June 8-10, 2011. Proceedings*, chapter Mobile Augmented Reality Based on the Semantic Web Applied to Ambient Assisted Living, pages 17–24. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.
[65] Id Software. *The Quake II's MD2 file format*. http://tfc.duke.free.fr/old/models/md2.htm, 1997.
[66] Ecma International. Ecmascript language specification, 2015.
[67] International Organization for Standardization (ISO). *Representations of dates and times*. https://www.w3.org/TR/xmlschema-2/, 1988.
[68] Itseez. Open source computer vision (opencv), 2016.
[69] Simon Josefsson. The base16, base32, and base64 data encodings. 2006.
[70] Hirokazu Kato and Mark Billinghurst. Artoolkit library, 1999.
[71] Hirokazu Kato and Mark Billinghurst. Marker tracking and hmd calibration for a video-based augmented reality conferencing system. In *Augmented Reality, 1999.(IWAR'99) Proceedings. 2nd IEEE and ACM International Workshop on*, pages 85–94. IEEE, 1999.
[72] Khronos Group. The standard for embedded accelerated 3d graphics, 2017.
[73] Kickstarter. *URL: https://www.kickstarter.com/*.
[74] I. Klabukov and A. Ostanin. *Hudway Glass, URL: http://hudwayglass.com/*.
[75] George Koutromanos and Georgios Styliaras. The buildings speak about our city: A location based augmented reality game. In *Information, Intelligence, Systems and Applications (IISA), 2015 6th International Conference on*, pages 1–6. IEEE, 2015.
[76] Max Krichenbauer, Goshiro Yamamoto, Takafumi Taketomi, Christian Sandor, and Hirokazu Kato. [demo] towards augmented reality user interfaces in 3d media production. In *Mixed and Augmented Reality (ISMAR), 2014 IEEE International Symposium on*, pages 351–351, Sept 2014.
[77] Max Krichenbauer, Goshiro Yamamoto, Takafumi Taketomi, Christian Sandor, and Hirokazu Kato. Towards augmented reality user interfaces in 3d media production. In *Mixed and Augmented Reality (ISMAR), 2014 IEEE International Symposium on*, pages 23–28, Sept 2014.
[78] Daniela Kugelmann, Leonard Stratmann, Nils Nühlen, Felix Bork, Saskia Hoffmann, Golbarg Samarbarksh, Anna Pferschy, Anna Maria von der Heide, Andreas Eimannsberger, Pascal Fallavollita, Nassir Navab, and Jens Waschke. An augmented reality magic mirror as additive teaching device for gross anatomy. *Annals of Anatomy - Anatomischer Anzeiger*, 2017.
[79] Allana G. LeBlanc and Jean-Philippe Chaput. Pokémon go: A game changer for the physical inactivity crisis? *Preventive Medicine*, 101(Supplement C):235 – 237, 2017.
[80] Martin Lechner. Arml 1.0 - augmented reality markup language, 2009.
[81] F. Ledermann and D. Schmalstieg. April: a high-level framework for creating augmented reality presentations. In *Virtual Reality, 2005. Proceedings. VR 2005. IEEE*, pages 187–194, March 2005.
[82] Gun A Lee and Mark Billinghurst. A component based framework for mobile outdoor ar applications. In *Proceedings of the 12th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and Its Applications in Industry*, pages 207–210. ACM, 2013.
[83] Gun A Lee, Andreas Dunser, Seungwon Kim, and Mark Billinghurst. Cityviewar: A mobile outdoor ar application for city visualization. In *Mixed and Augmented Reality (ISMAR-AMH), 2012 IEEE International Symposium on*, pages 57–64. IEEE, 2012.

[84] Zoopla Property Group Limited. Zoopla property search, 2015.

[85] Stephan Lukosch, Mark Billinghurst, Leila Alem, and Kiyoshi Kiyokawa. Collaboration in augmented reality. *Computer Supported Cooperative Work (CSCW)*, pages 1–11, 2015.

[86] B. MacIntyre, H. Rouzati, and M. Lechner. Walled gardens: Apps and data as barriers to augmenting reality. *IEEE Comput. Graph. Appl.*, 33(3):77–81, July 2013.

[87] Blair MacIntyre, Alex Hill, Hafez Rouzati, Maribeth Gandy, and Brian Davidson. The argon ar web browser and standards-based ar application environment. In *Mixed and Augmented Reality (ISMAR), 2011 10th IEEE International Symposium on*, pages 65–74. IEEE, 2011.

[88] Jacob B. Madsen and Claus B. Madsen. Handheld visual representation of a castle chapel ruin. *J. Comput. Cult. Herit.*, 9(1):6:1–6:18, November 2015.

[89] Tamás Matuszka, Gergő Gombos, and Attila Kiss. *Virtual Augmented and Mixed Reality. Designing and Developing Augmented and Virtual Environments: 5th International Conference, VAMR 2013, Held as Part of HCI International 2013, Las Vegas, NV, USA, July 21-26, 2013, Proceedings, Part I*, chapter A New Approach for Indoor Navigation Using Semantic Webtechnologies and Augmented Reality, pages 202–210. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[90] Tamás Matuszka, Sándor Kámán, and Attila Kiss. *Virtual, Augmented and Mixed Reality. Designing and Developing Virtual and Augmented Environments: 6th International Conference, VAMR 2014, Held as Part of HCI International 2014, Heraklion, Crete, Greece, June 22-27, 2014, Proceedings, Part I*, chapter A Semantically Enriched Augmented Reality Browser, pages 375–384. Springer International Publishing, Cham, 2014.

[91] Tamás Matuszka and Attila Kiss. Alive cemeteries with augmented reality and semantic web technologies. *International Journal of Computer, Information Science and Engineering*, 8:32–36, 2014.

[92] Spot Metrix. 3dar, 2015.

[93] Microsoft Corporation. *Microsoft Remote Assist*. https://www.youtube.com/watch?v=UpmolMrf5HQ, 2018.

[94] Paul Milgram, Haruo Takemura, Akira Utsumi, and Fumio Kishino. Augmented reality: a class of displays on the reality-virtuality continuum. volume 2351, pages 282–292, 1995.

[95] David Molyneaux and Selim BenHimane. "it's a pirate's life" ar game. In *Mixed and Augmented Reality (ISMAR), 2014 IEEE International Symposium on*, pages 361–362. IEEE, 2014.

[96] N. Navab, T. Blum, Lejing Wang, A. Okur, and Thomas Wendler. First deployments of augmented reality in operating rooms. *Computer*, 45(7):48–55, July 2012.

[97] N. Navab, S.-M. Heining, and J. Traub. Camera augmented mobile c-arm (camc): Calibration, accuracy study, and clinical applications. *Medical Imaging, IEEE Transactions on*, 29(7):1412–1423, July 2010.

[98] Jakob Nielsen. *Usability Engineering*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[99] Nintendo Co., Ltd. Pokemon go.

[100] Lyndon JB Nixon, Jens Grubert, Gerhard Reitmayr, and James Scicluna. Smartreality: Integrating the web into augmented reality. In *I-SEMANTICS (Posters & Demos)*, pages 48–54. Citeseer, 2012.

[101] Open Geospatial Consortium OGC. Geography markup language encoding standard (gml). On-line, 2012.

[102] Open Geospatial Consortium OGC. Geography markup language ontology. On-line, 2012.

[103] Open Geospatial Consortium OGC. Augmented reality markup language 2.0 arml 2.0. On-line, 2015.

[104] Object Management Group (OMG). Xml metadata interchange (xmi®), 2015.

[105] Francisco Pereira, Catarina Silva, and Mário Alves. Virtual fitting room augmented reality techniques for e-commerce. In MariaManuela Cruz-Cunha, João Varajão, Philip Powell, and Ricardo Martinho, editors, *ENTERprise Information Systems*, volume 220 of *Communications in Computer and Information Science*, pages 62–71. Springer Berlin Heidelberg, 2011.

[106] T. Piumsomboon, D. Altimira, H. Kim, A. Clark, Gun Lee, and M. Billinghurst. Grasp-shell vs gesture-speech: A comparison of direct and indirect natural interaction techniques in augmented reality. In *Mixed and Augmented Reality (ISMAR), 2014 IEEE International Symposium on*, pages 73–82, Sept 2014.

[107] Thammathip Piumsomboon, Adrian Clark, and Mark Billinghurst. [demo] g-siar: Gesture-speech interface for augmented reality. In *Mixed and Augmented Reality (ISMAR), 2014 IEEE International Symposium on*, pages 365–366. IEEE, 2014.

[108] Pivotal Software. *Spring Framework*. http://www.spring.io/, 2017.

[109] Augmented Pixels. Zoopla property search, 2015.

[110] PTC Inc. *Vuforia Augmented Reality SDK*. https://www.qualcomm.com/products/vuforia, 2012.

[111] H. Regenbrecht, T. Lum, P. Kohler, C. Ott, M. Wagner, W. Wilke, and E. Mueller. Using augmented virtuality for remote collaboration. *Presence: Teleoper. Virtual Environ.*, 13(3):338–354, July 2004.

[112] Gerhard Reitmayr. A list of augmented reality sdks and apis.

[113] Gerhard Reitmayr and Dieter Schmalstieg. Semantic world models for ubiquitous augmented reality. 2005.

[114] Vinny Reynolds, Michael Hausenblas, Axel Polleres, Manfred Hauswirth, and Vinod Hegde. Exploiting linked open data for mobile augmented reality. In *W3C Workshop: Augmented Reality on the Web*, volume 1. June, 2010.

[115] Dariusz Rumiński. An experimental study of spatial sound usefulness in searching and navigating through ar environments. *Virtual Reality*, 19:223–233, 2015.

[116] Dariusz Rumiński. An experimental study of spatial sound usefulness in searching and navigating through ar environments. *Virtual Reality*, 19(3-4):223–233, 2015.

[117] Dariusz Rumiński, Mikołaj Maik, and Krzysztof Walczak. Mixed reality stock trading visualization system. In *International Conference on Augmented Reality, Virtual Reality and Computer Graphics*, pages 301–310. Springer, Cham, 2018.

[118] Dariusz Rumiński and Krzysztof Walczak. Creation of interactive ar content on mobile devices. In *Proc. of International Conf. on Business Information Systems*, pages 258–269. Springer, 2013.

[119] Dariusz Rumiński and Krzysztof Walczak. Creation of interactive ar content on mobile devices. volume 160, pages 258–269. 2013.

[120] Dariusz Rumiński and Krzysztof Walczak. Carl: A language for modelling contextual augmented reality environments. In Luis M. Camarinha-Matos, Nuno S. Barrento, and Ricardo Mendonça, editors, *Technological Innovation for Collective Awareness Systems, in: IFIP Advances in Information and Communication Technology*, volume 432, pages 183–190. Springer, 2014.

[121] Dariusz Rumiński and Krzysztof Walczak. Dynamic composition of interactive ar scenes with the carl language. In *The 5th International Conf. on Information, Intelligence, Systems and Applications*, pages 329–334. IEEE, 2014.

[122] Dariusz Rumiński and Krzysztof Walczak. Semantic model for distributed augmented reality services. In *Proceedings of the 22Nd International Conference on 3D Web Technology*, Web3D '17, pages 13:1–13:9, New York, NY, USA, 2017. ACM.

[123] Dariusz Rumiński and Krzysztof Walczak. An architecture for distributed semantic augmented reality services. In *Proceedings of the 23rd International ACM Conference on 3D Web Technology*, page 18. ACM, 2018.

[124] Michele Ruta, Floriano Scioscia, Danilo De Filippis, Saverio Ieva, Mario Binetti, and Eugenio Di Sciascio. A semantic-enhanced augmented reality tool for openstreetmap poi discovery. *Transportation Research Procedia*, 3:479–488, 2014.

[125] Joan Puig Sanz. Beyondar, 2015.

[126] Dieter Schmalstieg, Anton Fuhrmann, Gerd Hesina, Zsolt Szalavári, L Miguel Encarnaçao, Michael Gervautz, and Werner Purgathofer. The studierstube augmented reality project. *Presence: Teleoperators and Virtual Environments*, 11(1):33–54, 2002.

[127] Dieter Schmalstieg and Gerhard Reitmayr. The world as a user interface: Augmented reality for ubiquitous computing. In *Location based services and telecartography*, pages 369–391. Springer, 2007.

[128] H. Seichter, J. Looser, and M. Billinghurst. Composar: An intuitive tool for authoring ar applications. In *The 7th IEEE/ACM International Symp. on Mixed and Augmented Reality (ISMAR 2008)*, pages 147–148. IEEE Xplore, 2008.

[129] Peter Sevcik. Defining the application performance index. *Business Communications Review*, pages 8–10, 2005.

[130] Augmented Reality Lab S.L. Ar browser framework, 2014.
[131] Aduna Software. Sesame rdf query language. On-line, 2001.
[132] Gheric Speiginer, Blair MacIntyre, Jay Bolter, Hafez Rouzati, Amy Lambeth, Laura Levy, Laurie Baird, Maribeth Gandy, Matt Sanders, Brian Davidson, Maria Engberg, Russ Clark, and Elizabeth Mynatt. *Human-Computer Interaction: Users and Contexts: 17th International Conference, HCI International 2015, Los Angeles, CA, USA, August 2-7, 2015, Proceedings, Part III*, chapter The Evolution of the Argon Web Framework Through Its Use Creating Cultural Heritage and Community–Based Augmented Reality Applications, pages 112–124. Springer International Publishing, Cham, 2015.
[133] D. Stanimirovic, N. Damasky, S. Webel, D. Koriath, A. Spillner, and D. Kurz. A mobile augmented reality system to assist auto mechanics. In *Mixed and Augmented Reality (ISMAR), 2014 IEEE International Symposium on*, pages 305–306, Sept 2014.
[134] Philipp Stefan, Séverine Habert, Alexander Winkler, Marc Lazarovici, Julian Fürmetz, Ulrich Eck, and Nassir Navab. *A Mixed-Reality Approach to Radiation-Free Training of C-arm Based Surgery*, pages 540–547. Springer International Publishing, Cham, 2017.
[135] William Steptoe. Ar-rift: Stereo camera for the rift & immersive ar showcase. oculus developer forums, https://developer.oculusvr.com/forums/viewtopic.php?f=28&t=5215, 2013.
[136] Ivan E Sutherland. The ultimate display. *Multimedia: From Wagner to virtual reality*, 1965.
[137] Ivan E. Sutherland. A head-mounted three dimensional display. In *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*, AFIPS '68 (Fall, part I), pages 757–764, New York, NY, USA, 1968. ACM.
[138] T-immersion inc. *Total Immersion's D'Fusion Studio suite*. https://www.youtube.com/watch?v=2QQTf4oX_uE.
[139] Unity Technologies. Unity 3d, 2016.
[140] Wavefront Technologies. The wavefront obj format, 1984.
[141] The World Wide Web Consortium (W3C). *International Resource Identifier (IRI)*. https://www.w3.org/International/articles/idn-and-iri/.
[142] The World Wide Web Consortium (W3C). *Universal Resource Identifier (URI)*. https://www.w3.org/wiki/URI.
[143] Bruce H Thomas. A survey of visual mixed and augmented reality gaming. *Computers in Entertainment (CIE)*, 10(3):3, 2012.
[144] Van Velsen, Martin and Fercoq, Robin. *3ds specification*. http://www.martinreddy.net/gfx/3d/3DS.spec, 1997.
[145] Silviu Vert and Radu Vasiu. Integrating linked open data in mobile augmented reality applications-a case study. *TEM Journal*, 4(1):35–43, 2015.
[146] The World Wide Web Consortium (W3C). The resource description framework (rdf).
[147] The World Wide Web Consortium W3C. Resource description framework (rdf). On-line, 2014.
[148] World Wide Web Consortium (W3C). RDF 1.1 XML Syntax.
[149] World Wide Web Consortium W3C. Rdql - a query language for rdf. On-line, 2004.
[150] World Wide Web Consortium W3C. The web ontology language 2 (owl2). On-line, 2012.
[151] World Wide Web Consortium W3C. The web ontology language (owl). On-line, 2012.
[152] World Wide Web Consortium W3C. Simple protocol and rdf query languaue (sparql). On-line, 2013.
[153] World Wide Web Consortium W3C. Rdf schema 1.1. On-line, 2014.
[154] World Wide Web Consortium (W3C). *The Web Ontology Language (OWL)*, accessed 2016-04-07.
[155] K. Walczak. Beh-vr: modelling behavior of dynamic virtual reality contents. In *Interactive Technologies and Sociotechnical Systems LNCS 4270*, pages 40–51. Springer-Verlag, Heidelberg, 2006.
[156] Krzysztof Walczak, Wojciech Cellary, and Martin White. Virtual museum exbibitions. *Computer*, 39(3):93–95, 2006.
[157] Krzysztof Walczak, Dariusz Rumiński, and Jakub Flotyński. Building contextual augmented reality environments with semantics. In *Virtual Systems Multimedia (VSMM)*, pages 353–361, Dec 2014.
[158] Krzysztof Walczak, Wojciech Wiza, Rafał Wojciechowski, Adam Wójtowicz, Dariusz Rumiński, and Wojciech Cellary. *Building Augmented Reality Presentations with Web 2.0 Tools*, pages 595–605. Springer International Publishing, Cham, 2015.

[159] Y. Wang, T. Langlotz, M. Billinghurst, and T. Bell. An authoring tool for mobile phone ar environments. In *Proc. of the New Zealand Computer Science Research Student Conf.*, pages 1–4, 2009.

[160] R. Wojciechowski. *Modeling interactive educational scenarios in a mixed reality environment.* Doctoral Dissertation. Technical University of Gdańsk, 2008.

[161] Rafał Wojciechowski and Wojciech Cellary. Evaluation of learners' attitude toward learning in aries augmented reality environments. *Comput. Educ.*, 68:570–585, October 2013.

[162] Rafał 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*, pages 135–144. ACM, 2004.

[163] World Wide Web Consortium (W3C). A JSON-based Serialization for Linked Data.

[164] World Wide Web Consortium (W3C). Notation3 (N3): A readable RDF syntax.

[165] World Wide Web Consortium (W3C). RDF 1.1 N-Triples.

[166] World Wide Web Consortium (W3C). W3c semantic web activity.

[167] Stefan Zander, Chris Chiu, and Gerhard Sageder. A computational model for the integration of linked data in mobile augmented reality applications. In *Proceedings of the 8th International Conference on Semantic Systems*, I-SEMANTICS '12, pages 133–140, New York, NY, USA, 2012. ACM.

[168] J. Zauner and M. Haller. Authoring of mixed reality applications including multi-marker calibration for mobile devices. In *In: Proc. of the 10th Eurographics Symp. Virtual Env. (EGVE)*, pages 87–90, 2004.

[169] Zhiying Zhou, Adrian David Cheok, Xubo Yang, and Yan Qiu. An experimental study on the role of 3d sound in augmented reality environment. *Interacting with Computers*, 16(6):1043 – 1068, 2004.

Glossary

AR – Augmented Reality
ARP – AR Service Provider
ARR – AR Resource
CARE – Contextual Augmented Reality Environment
CARL – Contextual Augmented Reality Language
CARP – Contextual Augmented Reality Presentation
CUC – Contextual User Context
dCARP – Description of Contextual Augmented Reality Presentation
GML – Geography Markup Language Encoding Standard
IRI – Internationalized Resource Identifier
KB – Knowledge Base
RDF – Resource Description Framework
RDFS – Resource Description Framework Schema
RV – The Reality-Virtuality continuum
SARO – Semantic Augmented Reality Ontology
SDMM – Semantic Discovery and Matching Method
SPARQL – Protocol And RDF Query Language
VR – Virtual Reality
W3C – World Wide Web Consortium

List of Figures

List of Tables

Listings