

SEMANTIC WEB INTERIOR DECORATION WITH QUERIES AND 3D
REPRESENTATION OVER WEBRTC AND X3DOM

by

MALVINA STEIAKAKI

Bachelor Honours Degree in Applied Information Technology & Multimedia, Technological
Educational Institute of Crete, 2013

A THESIS

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

DEPARTMENT OF INFORMATICS ENGINEERING

SCHOOL OF APPLIED TECHNOLOGY

TECHNOLOGICAL EDUCATIONAL INSTITUTE OF CRETE

2016

Approved by:

Major Professor
Dr. Athanasios G. Malamos

Copyright © 2016

MALVINA STEIAKAKI

All rights reserved. No part of this material may be reproduced, displayed, modified or distributed without prior written permission from the author or author's institute- except in the case of noncommercial research and nonprofit education which has to be accompanied by the appropriate citation.

Abstract

In this thesis, we present a web based framework for interior decoration that merges Real Time Communication, Web3D technologies and Semantic Web ontologies. The system based on an OWL framework which describes the interior design concepts. A questionnaire with a specific number of questions regarding to various parameters of the room space are going to be filled by users. These choices are SPARQL queries which allow users to query the ontology for instances that are going to be matching their needs. After this, the system is responding by displaying the appropriate decoration solutions automatically to graphical layout via WebRTC with X3DOM format according to their ontological descriptors. The X3DOM GUI produces automatic visual reconstruction of a design scheme in a browser and the users can modify the scene according to their desires. The users have the ability to modify the physical and spatial characteristics of 3D object. Such capabilities are updated in real-time in each user's browser with the assistance of the MCU while at the same time, all users communicate each other with video conferencing.

Keywords: Semantic Web, X3D, X3DOM, SPARQL, WebRTC, SPARQL

Table of Contents

Copyright © 2016	ii
Abstract	iii
Table of Contents	1
List of Figures	4
List of Tables	6
Acknowledgements	7
Dedication	8
Preface	9
Chapter 1 - Introduction	10
Chapter 2 - Background & Motivation	12
Chapter 3 - Semantic Web	15
3.1 Web Ontology Language (OWL)	15
3.1.1 Basics elements of OWL	17
3.2 Simple Protocol and RDF Query Language (SPARQL)	18
3.3 Jena Framework	19
3.3.1 Jena SDB	19
3.4 Related Work	20
3.4.1 Semantic Web solutions	20
3.4.2 SPARQL solutions	22
3.5 DECoration Ontology (DECO)	24
3.6 DECO Web-based Ontology Editor	32
Chapter 4 - Web3D	35
4.1 X3D Standard	35
4.2 X3DOM	36
4.3 Build an X3D file	37
4.4 Setup and run X3DOM	39
4.5 Related Work	42
4.5.1 X3D Visualization solutions	42

4.6 DECO Web Editor	43
Chapter 5 - Web Real-Time Communications (WebRTC).....	46
5.1 WebRTC Network Protocols	48
5.2 WebRTC Network Topologies	49
5.3 Related Work	50
5.3.1 WebRTC solutions.....	50
5.4 Our WebRTC approach	51
Chapter 6 - Architecture and Prototype	53
6.1 Functional Design	53
6.1.1 Login and Registration Manager.....	53
6.1.2 “Session” Manager.....	55
6.1.3 Content Manager.....	57
6.1.4 Collaborative Design Manager	58
6.2 Technical Design	63
6.2.1 Login and Registration Manager.....	63
6.2.2 Session Manager	65
6.2.3 Content Manager.....	65
Web Ontology Editor Module	65
Semantics	66
6.2.4 Collaborative Design Manager	67
MCU - Real Time Communication.....	67
Collaborative Web3D	67
Multi- Conferencing Module	69
Chapter 7 - Conclusions.....	74
References.....	76
Appendix A - WEBRTC Configuration and programming.....	81
Configuring Node.js & Socket.io.....	81
Creation of the web server	82
Generating Certificates with OpenSSL Toolkit.....	83
Programming the server	87
Integration of a client-side	88

Appendix B - Glossary of Technical Terms 95

List of Figures

Figure 1-1. Our proposed framework	11
Figure 2-1. Examples of online applications	12
Figure 2-2. Multiuser real-time system for 3D representation	13
Figure 3-1. OWL sublanguages	15
Figure 3-2. DECO Classes	25
Figure 3-3. DECO ontology – graphical layout.....	26
Figure 3-4. DECO object properties	27
Figure 3-5. DECO datatype properties	30
Figure 3-6. DECO architecture (A) Content Provider, (B) SPARQL mechanism	34
Figure 4-1. X3D Architecture	36
Figure 4-2. Examples of X3D visual representations	38
Figure 4-3. X3D rendering in browser.....	40
Figure 4-4. X3DOM debug mode for position coordinates	41
Figure 4-5. X3DOM debug mode for viewpoint	41
Figure 4-6. A set of living rooms which have created for this thesis	45
Figure 5-1. The history of communication protocols until WebRTC.....	46
Figure 5-2. Multipurpose WebRTC system.....	47
Figure 6-1. Login and Registration layout.....	54
Figure 6-2. Session process.....	56
Figure 6-3. Notification messages for user	56
Figure 6-4. Questionnaire and pop-up windows for 3D representation.....	57
Figure 6-5. DECO application layout	58
Figure 6-6. DECO Multi-user chat module. Left: tablet view, Right: desktop view	59
Figure 6-7. DECO Multi-user video call	59
Figure 6-8. Changing the color of an item in X3DOM scene.....	60
Figure 6-9. DECO tools	61
Figure 6-10. Inserting and uploading 3D items	61
Figure 6-11. Saving a picture in DECO framework	62
Figure 6-12. Saving a picture in a tablet device.....	62

Figure 6-13. Saving the X3D code in a database with DECO.....	62
Figure 6-14. Login and Registration System diagram.....	64
Figure 6-15. DECO architecture for semantic storing.....	66
Figure 6-16. DECO architecture for SPARQL engine.....	67
Figure 6-17. Process for real-time communication.....	69
Figure 6-18. Process for multi-user conferencing using our MCU.....	71
Figure 7-1. Cross-platform and multi-device support.....	74
Figure 7-2. Mapping an X3D scene to a real-world room space.....	75

List of Tables

Table 3-1. Datatype properties values	18
Table 3-2. SPARQL query variations	19
Table 4-1. The creation of a “box” with X3D	38
Table 4-2. Code for inserting X3D in HTML page	40
Table 4-3. Add an external file to X3DOM scene	42
Table 4-4. Adding listener to X3DOM nodes.....	44
Table 4-5. JavaScript code for changing color in X3DOM node	45
Table 6-1. Fuctionalities of DECO application	55
Table 6-2. Table users in MySQL.....	63
Table 6-3. Table ‘booths’ in MySQL	65
Table 6-4. Event for changing color from MCU to connected clients.....	68
Table 6-5. Listening of ‘changeColor’ event from clients.....	68
Table 6-6. MCU code for managing events for multi-user conferencing.....	70
Table 6-7. Clients code for managing the events for multi-user conferencing.....	71
Table 6-8. Code for STUN and addstream function	73

Acknowledgements

At this point, I would like to express my gratitude to my supervisor Dr. Athanasios G. Malamos, Associate Professor in the Dept. of Informatics Engineering of the Technological Educational Institute of Crete and Head of Multimedia Content Laboratory, for his comments and the guidance throughout the development process of this master thesis. The accomplishment of this work would not be possible without his helpful guidance and contribution. I would also like to thank my close friends and laboratory colleagues, who supported me all the way during this thesis. Finally, a special thanks to my family that supported me all these years of my studies.

Dedication

I would like to dedicate this thesis to my family for supporting and encouraging me all these years, being the reason of what I have achieved and become today. Last but not least, I would like to dedicate this thesis to my friends.

Preface

This report is my master thesis for the conclusion of my Master program in Informatics and Multimedia, Department of Informatics Engineering, School of Applied Technologies, Technological Educational Institute of Crete.

Having worked in area of the semantics before, I have become familiar with the current state of the art Semantic Web. Also, I have worked with the web technologies like HTML and JavaScript. In this thesis, the majority of these implementations focused into the efficient semantic representation of specific aspects via WebRTC protocol and X3DOM framework.

Chapter 1 - Introduction

Today, web applications' field is dominated by various technologies that come with convenient cross-platform capabilities. Amongst them the most commonly used is no other than HTML5 -a markup language for the presentation of multimedia-rich content in the Web- whereas the last few years X3DOM framework and WebRTC were stand out from the remaining. X3DOM is an open-source JavaScript framework capable of integrating 3D content into any webpage without the use of plugins, while WebRTC is an innovative technology that allows Real Time Communication through several JavaScript APIs.

Furthermore, web applications incorporate key elements of Semantic Web vision, a series of technologies that help computers and humans to understand and process the data in the Web. Semantic Web aims to transform these data by integrating human language concepts and their relations in current information management scheme to improve organization and boost search capabilities. OWL language provides the necessary means to design and translate such content under semantic terminology, making possible the extraction of new meaningful information. OWL comes with three different sublanguages, where OWL-DL supports maximum expressiveness while retaining computational completeness. Semantic Web languages like OWL are heavily depending on query languages for the execution of queries and extraction of information via HTTP requests. SPARQL is the most widely known query language for exploiting the information found in relational databases, making use of typical tables filled with RDF triples.

In this thesis, we present a web-based framework for the interior decoration of a room space, which successfully merges Real Time Communication, Web3D and Semantic Web technologies. The interior design concepts of the system are based on an OWL ontology which is hosted by Apache Jena Semantic Framework, while a questionnaire with specific number of questions regarding various parameters of the room space is filled by system's end users. This questionnaire's choices are coming from the execution of diverse SPARQL queries upon the underlying ontology, enabling in this way the correlation of user's needs (companies or consumers) with specific only instances of this ontology. The combination of these instances leads to the display of the appropriate decoration solutions automatically, with the assistance of WebRTC protocol as the communication mean and an X3DOM scene as the graphical layout according to their ontological descriptors. Moreover, the implemented GUI not only supports an

automatic visual reconstruction of a design scheme in user’s browser, but it also allows them to modify the scene according to their desires. Users have the ability to change the color of a specific item in the scene, rotate, scale, and move it inside the room space according to its x, y, z coordinates. Finally, the user of the system has the ability to ‘drag and drop’ additional items that have been exported by SPARQL responses, where each one shares something in common with user’s choices. Finally, the implemented application makes feasible the communication of an unrestricted number of users in each active group, using either video call or chat messaging services with the assistance of our own MCU server.

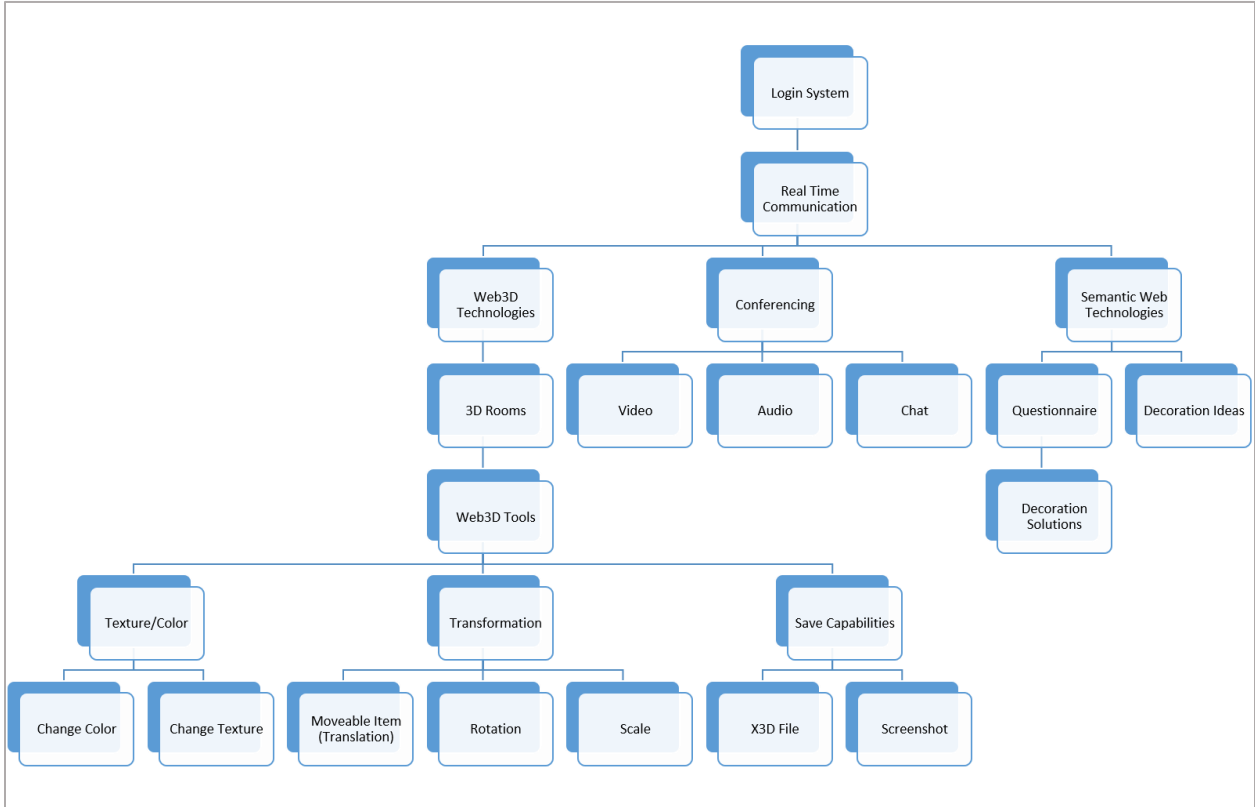


Figure 1-1. Our proposed framework

Chapter 2 - Background & Motivation

Seeking efficient decoration and interior design solutions for the needs of our application, we carried out a thorough search on the Internet for programs and tools dealing with such domains. The conducted search pointed out that the majority of these applications provide visual representation and modification capabilities according to user's desires. However, despite the fact that their visualization motive can be easily understood by the users and can be enriched with specific content and attributes (furniture, materials, colors, etc.), their overall functionality requires the installation of a custom-made browser plugin or additional software programs.



Figure 2-1. Examples of online applications

To be more specific, we will refer to some examples of applications that come across in the online market. The Autodesk HomeStyles [1], Roomsketcher [2] and Planner 5D [3] give the possibility of the representation in 2 and 3 dimensions and also provide options to change materials and color. The negative point in these applications is that implemented with the assistance of the Flash Player, which should be preinstalled on the user's computer. The previous also provides the ability to store the designed interior space but in such format that resembles the JSON format but unfortunately cannot be used elsewhere by the user, but only within the application. Another application that we found in our search is the Roomstyler [4] which provides 2D representation of an interior space with the ability of "Screenshot" in three dimensions (3D). Another application is the IKEA home planner [5] which to operate should establish its own plugin in the browser. Also, you can see Chapter 4, for more related work.

Within realization of this thesis we wanted to extend the experience of creating and modifying the interior by adding parts of semantic web and communication between client and decorator. We focused in the public that looking for complete solution to room visualization, so after the communication with experts in this area, can obtain a 3D representation of his interior space which would come from his desires. The real-time communication and interaction in three-dimensional interior design space without installation of any program classifies our application as alternative and new solution in the field of the interior decoration.

The main purpose of this work was the implementation of a multiuser real-time communication system for the decoration of room spaces unifying diverse Web-based technologies.

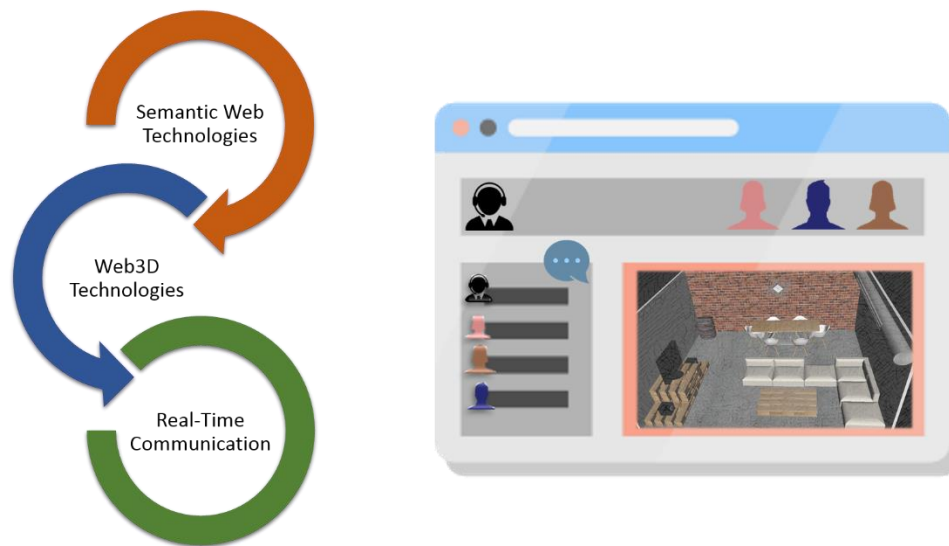


Figure 2-2. Multiuser real-time system for 3D representation

Our previous avocation with ontologies and Semantic Web gave birth to the idea of integrating the visual representation and semantic layers with a real-time communication layer between users. In this way, it could be created a standalone product as the basis for decorators and interior design companies, where the last ones could possibly develop it even further. Taking into consideration modern era's capabilities and technologies along with the continuous increment of users navigated to Internet via their mobile devices (e.g. smartphones, tablets, etc.), our work involved innovative technologies that are independent of installing plugins or additional software.

Moreover, the Responsive Web Design was applied throughout our application in order to ensure its ease of use and accessibility from any kind of device and platform.

Concerning the three-dimensional representation aspect, we chose to take advantage of the X3DOM technology, an innovative plugin-less framework capable of displaying 3D content on common web browsers. It is based on a string-based layout that is closely related to the XML format, where an entire X3D scene can be either integrated to a simple HTML page, or consolidated by an external file through the “Inline” statement of X3DOM API. Moreover, it supports the real-time movement, coloring and resizing of any physical object in the 3D space, features that were deemed critical for the creation of an interactive collaborative environment, rather than a typical visual representation tool.

The semantic layer of the application is based on DECO framework, a titular research project which gave birth to an interior design and decoration ontology developed by the Multimedia Content Laboratory of TEI of Crete. Its web-based interface was implemented with the assistance of Apache Jena and an SDB triple-store module, enabling in this way its functionality as an online ontological editor with capabilities similar to these of the Protégé open-source software. The usage of this framework by content providers will make easier the storage of their room spaces and objects as ontological concepts, which can be later queried via SPARQL language and being displayed according to users’ desires.

Our vision to combine all of the aforementioned features and technologies into a single real-time web-based application led us to the espousal of WebRTC. This technology allows interconnected users of a specific website to create groups in order to interact with each other. Our application enhances this interaction by providing video streams, text messages and the sharing of a common 3D space between the participating users. Last but not least, besides these characteristics the implemented system is governed by a set of “roles”, which are responsible for the readjustment of content depending on the logged user role.

In the following chapters, we present the general theoretic background about the technologies and standards that used for the creation of our framework for interior decoration and our own MCU server which manages and emits the messages received from and to connected users.

Chapter 3 - Semantic Web

3.1 Web Ontology Language (OWL)

The Web Ontology Language (OWL) is a language specially made for the Semantic Web, in order to sufficiently describe concepts stemming from a specific domain and organize them in a hierarchical manner. This organization takes place in such a way that primarily aids computer software to understand these concepts, rather than presenting this information to humans. Over and above, OWL constitutes an active part of W3C's Semantic Web technology stack, which is comprised of RDF, RDFS, SPARQL and other languages. OWL consists of the three sublanguages that are displayed in the following figure and are shortly described below:



Figure 3-1. OWL sublanguages

- ❖ **OWL Lite** intends to support users who seek for a classification hierarchy with simple constraints. This sublanguage of OWL provides a tool for the fast translation and migration of domain's concepts to taxonomies known as thesauri. Moreover, OWL Lite comes with lower

complexity compared to the upcoming OWL-DL sublanguage, by excluding arbitrary cardinality restrictions, disjointness statements and enumerated classes.

❖ **OWL-DL** intends to support users who want maximum expressiveness without sacrificing the computational completeness of the practical reasoning algorithms and their decidability [6]. OWL-DL includes all OWL language constructs with several limitations. These limitations are listed below:

- Explicit typing
- Vocabulary partitioning: Any resource is allowed to be exclusively a part of built-in vocabulary and not more than one of these (for instance, a class cannot also be at the same time an individual)
- Properties distinction: Object and datatype properties are defined as disjoint constructs
- No transitive cardinality restrictions are supported.

This sublanguage's name resulted from its close relationship with Description Logics domain. After years of research in DL languages, a well-defined semantic vocabulary with a sufficient number of properties was composed for the sake of OWL language. Furthermore, systems that adopt this kind of knowledge representation make use of well-known reasoning algorithms and are able to form easier relationships with other formal logics, like Fuzzy, First-Order or Modal Logics.

❖ **OWL Full** bears different semantics compared to OWL Lite and OWL-DL sublanguages, since it was designed to support the entire set of constructors. It provides a maximum expressiveness which can be used in any axioms' combination, as long as the result is a legal RDF structure. For example, a class defined in OWL Full can concurrently be a collection of individuals and an individual itself. Therefore, the advantage of OWL Full lies to its full compatibility with RDF format, both semantically and syntactically. However, the latter also reveals its main disadvantage, which is no other than the fact that there is no reasoning system that can successfully support complete reasoning for every single feature of OWL Full sublanguage.

3.1.1 Basics elements of OWL

The **Class** element, which is used to assort resources that share common ground and is defined using the owl:Class construct. Classes unrelated to each other are marked with the owl:disjointWith construct, while classes that share similar properties are declared with the owl:equivalentClass construct. There are also two predefined classes for each OWL ontology, owl:Thing and owl:Nothing, where the first one contains all user defined classes and the second is an empty instance of the Class element. So, every class is a subclass of owl:Thing and superclass of owl:Nothing.

The grouping capabilities of Class element leads to the creation of several sets of members, known as **Individuals** that come with additional property characteristics. **Properties** are another basic element of OWL language, used to describe semantical relationships between two sets of individuals.

OWL defines three kind of **properties**, namely Object, Datatype and Annotation. Object properties are used to define relations between the individuals of an ontology, while datatype properties relate individuals to RDF Literals, XML Schemas, and other datatypes. Table 3-1 below displays all the possible values that a datatype property is allowed to own.

xsd:string	xsd:time
xsd:decimal	xsd:gMonthDay
xsd:integer	xsd:token
xsd:nonPositiveInteger	xsd:Name
xsd:long	xsd:Boolean
xsd:unsignedLong	xsd:double
xsd:hexBinary	xsd:positiveInteger
xsd:dateTime	xsd:short
xsd:gYear	xsd:unsignedShort
xsd:anyURI	xsd:date
xsd:NMTOKEN	xsd:gDay
xsd:normalizedString	xsd:language
xsd:float	xsd:NCName

xsd:nonNegativeInteger	xsd:byte
xsd:negativeInteger	xsd:unsignedByte
xsd:int	xsd:gYearMonth
xsd:unsignedInt	xsd:gMonth
xsd:base64Binary	rdfs:Literal

Table 3-1. Datatype properties values

On the other hand, Annotation properties are able to relate any kind of element to any kind of value. However, this type of property possess a stronger theoretical background, rather than a practical use to daily applications.

3.2 Simple Protocol and RDF Query Language (SPARQL)

SPARQL is a query language which standardized and became a W3C Recommendation with the version 1.1 in March 2013. It defines a data access protocol for use with RDF (Resource Description Framework) language. More specifically, SPARQL is a semantic query language for database management systems, capable to retrieve and manipulate data stored in RDF format.

The authoring of queries in SPARQL is feasible by any user, while its vocabulary has many similarities with the SQL language, due to the fact that the first shares common keywords from the second, such as the SELECT and WHERE. However, SPARQL introduced new keywords such as the OPTIONAL and FILTER for specific usage on semantic datasets. Any SPARQL query is based on a set of triples that consist of the subject, predicate and object components. The idea behind this mechanism is the matching of query triples with the existing RDF triples, in order to find the best suitable stored solution. Table 3-2 below presents the four different purposes' query variations.

Query Variation	Purpose
SELECT	Used to extract raw values from a SPARQL endpoint, the results are returned in a table format.
CONSTRUCT	Used to extract information from the SPARQL endpoint and transform the results into valid RDF.

ASK	Used to provide a simple True/False result for a query on a SPARQL endpoint.
DESCRIBE	Used to extract an RDF graph from the SPARQL endpoint, the content of which is left to the endpoint to decide based on what the maintainer deems as useful information.

Table 3-2. SPARQL query variations

3.3 Jena Framework

Jena is a widely known framework based on Java language, which has been used for the development of various Semantic Web applications. It is an open source project that provides several Java APIs for dealing with OWL, RDF, RDFS, SPARQL, and other standardized languages or reasoning engines. It works as an additional layer that translates the semantics into Java artifacts (classes, methods, attributes, etc.), supporting all three sublanguages (Full, Description Logic and Lite) of OWL and adding methods for the validation of ontologies via a fully functional OWL API. Concerning the persistence storage of semantical data, Jena supports both raw files and database backend solutions through an efficient triple-store mechanism. The SPARQL queries are applied to the knowledge graph generated by Jena's API, while simple GET or POST methods can be used to send and fetch responses from a HTTP Web Server. Finally, Jena supports various external reasoners (e.g. Pellet, Racer, FaCT) for the sufficient inference of OWL implementations and the deduction of additional knowledge, datum that cannot be derived from typical reasoning procedures.

All of the above show that Apache Jena Java Framework is not just an excellent tool for investing and developing Semantic Web applications, but it has already extensively contributed in the evolution of the World Wide Web into a web of data.

3.3.1 Jena SDB

Jena SDB is a native triple store component of Jena framework for the persistent storage and querying of RDF data using third party databases. Triple stores are custom-build databases that provide scalability and query-rich functionalities over RDF datasets. SDB belongs to DBMS category, which means that it is taking advantage of the storing and retrieval mechanisms granted by the underlying database system of the application. It provides a series of tools and

functionalities for the loading, backup, security and administration of semantic databases, while a SPARQL interface is placed on top of the classic database systems. Even though that there are numerous factors (e.g. hardware, dataset size, operating system, etc.) that can affect RDF's triple store performance, SDB is able to query millions of triples and load several thousands of them per second.

3.4 Related Work

3.4.1 Semantic Web solutions

The Semantic Web comes with the promise to change our daily life by taking advantage of the present information found on the Web. Its main goal is to add semantics to these hidden data in such a way that machines can both understand and reason about their content [7]. Today, ontologies and ontology-based systems appear as the nodal point for the development of potent and profitable Internet commerce solutions. Semantic Web has a much higher influence on e-commerce and e-work applications than the current Web, showing them the way to access a wide range of information about business, economics, analytics, etc. Such domains will be more prominent compared to others in the near future, since they have already received improved efficiency and better optimization of their data thanks to appliance of semantics.

In paper [8], authors demonstrated the practicability of ontologies within e-commerce applications, along with an indicative categorization according to their scope of use. In short, ontologies successfully achieved the categorization of products within catalogues, categorization of services and web-services, identification of unique products, and classification of industrial statistics. A notable work in paper [9] presented an Intelligent DIY (Do it Yourself) e-commerce system for vehicle design that was based on ontologies and three-dimensional (3D) visualization techniques. The implemented system aimed to provide a more realistic visualization of its products, by including an online product configuration tool backed by a three stepped OWL ontology:

- ❖ Insertion of object's 3D parts
- ❖ Selection of the desired texture
- ❖ Storing the 3D animation for future use

Staying on e-commerce domain and following the same pattern, the first electronic shops utilizing ontologies and visualization capabilities also made their appearance. Amongst them, a Configurable Electronic Shop Platform in the form of a web-based application [10] allowed users to assemble furniture parts of various products. These parts could be anytime stored in a specific system repository maintained by the manufacturer of the end product.

Leaving aside economical realm, several works in architecture and interior design disciplines have been presented the last few years. In [11] was developed a 3D interactive system for the semantical description of the physical relations between objects in a room. Its interior design concepts were contained into XML Topic Maps (XTM), giving users the ability to build a 2D design of a room which was afterwards translated to its relative Web3D model in VRML language. The overall system, however, was based on an Interactive Scene Viewer for the modification and movement of the objects, while the majority of its modules communicated with each other via SOAP messages. A more recent work [12] put forward an interesting proposal for the definition of a unified architectural environment. This environment provided a common building plan for all workmanships, where each one was able to visualize only these elements that correspond to its profession. The proposal was built around an OWL ontology that encompassed all aspects of the building process -from conceptual elements to design elements- and a set of ontological descriptions for the presentation of X3D models in a VR environment.

The same semantical and presentation languages were also used to compose DEC-O project [13], an ontological framework that aimed to organize all aspects of an interior space into a usable structure. This framework merged Web3D technologies with SVG graphics and Semantic Web ontologies. Its core rested upon an OWL-DL language that described various interior design concepts, from the structural elements of a room space down to the color of a 3D object (e.g. a furniture). Each object incorporated a 2D representation in SVG format, followed by its on-the-fly X3D representation thanks to an XSLT transformation algorithm. All these technologies were combined with a SWRL recommender system for the automated implication of the best fitting room space, according to a predefined set of interior design rules and the needs of the end user. Such data were collected by two separate wizards, with the first one to correspond to platform's managers and the second one to platform's consumers. The managers were able to modify the ontology through Jena Framework with the assistance of dynamically generated drop-down menus, add new individuals, modify existing ones, or add new statements to the current instance

of the ontology. On the other hand, consumers were able to query and search all the ontologies through an easy-to-answer questionnaire backed by an SDB Triple Store mechanism.

All of the aforementioned works showed that Web3D environments come with solid foundations on presentation and semantical annotation capabilities, but they still lack of reasoning mechanisms that can reveal extra information from graphics domain. For that reason, several multimodal interaction frameworks were soon composed to deal with this inefficiency. Some of them, as authors of the paper [14] used ontologies to merge spatial relationships between objects and their visual placement inside the space, allowing in this way users to interact with “placeable” objects and move them around as they like. Others as [15] proposed the use of Virtual Reality databases for graphics and physics simulations into an Artificial Intelligent knowledge base using semantic net representations. One more time, an ontology was responsible for describing objects with spatial attributes and storing the definition of spatial predicates. However, one of the latest attempts in this area introduced a framework capable of deducting spatial semantics based on an R-tree data structure and spatial reasoning techniques, without the use of a Semantic Web language or other ontological descriptors [16].

3.4.2 SPARQL solutions

The most important part of information retrieval and usability design in Semantic Web applications is their query formulation process. For that reason, many query languages have been developed for the sake of Semantic Web, with the majority of them to be applied in various e-shops. Even though that this kind of websites are composed of static content, product categories selected by the user are dynamically produced by SPARQL queries. The generated product’s description is based on the definition of the syntactic features of the language and their semantic interpretation according to each use-case scenario. However, triple store’s performance is one crucial concept that has to be taken into consideration during the development of the chosen semantic technology. In [17] was presented a novel approach for the improvement of relational databases performance through caching the results of SPARQL queries in triple stores. Their idea was based on the observation that large parts of a triple store usually do not change over time, but only a small part of query results is in fact affected by updates to the knowledge base.

Another work such as [18] described the method of content-based retrieval in Semantic Web, where the model was based on the axiom that similar models may partially share the same

semantics. The approach followed was a mixing of clustering and feedback methodologies for the correlation of the three dimensional models which were the individuals of an ontology. The proposed system was able to successfully return all the results to the user, where the irrelevant ones were skipped and user was pointed to the nearest cluster. The implemented system made use of OWL language and Jena Framework as its reasoning tools, while SPARQL [19] language was deployed to assist the overall query process. Such works pointed out a couple of standards in the topic of Semantic Web and more specifically into querying an ontology with SPARQL:

- ❖ Search all of the available products with multi-criteria search engines
- ❖ Group products into multi-level categories
- ❖ Make sales
- ❖ Update both the product catalogue and all items' availability
- ❖ Create/use shop baskets (end-users acting as buyers)
- ❖ Provide convenient pay methods with a secure e-payment credit card

SPARQL specification was also used for the implementation of Web-based graphical tools like NITELIGHT [20], a JavaScript written tool which merged various features for the creation of semantic queries in client side. These features included the browsing of an ontology in column view, an interactive graphical interface, a SPARQL-compliant visual query language with syntax viewer and the presentation of queries' results in user's browser.

Besides 3D modelling and presentation systems, SPARQL solutions can be found in automated reasoning and interior decoration areas. A mixing of AJAX, SVG, JSF, RDF, OWL, SPARQL, Jena technologies and DL reasoners gave birth to an automated reasoning system named SMART [21]. It stands for Semantic web information Management with Automated Reasoning Tool, and it was designed to synthesize and validate semantic queries, represent them through a useful graphical interface and map DL queries to SPARQL language. Its SPARQL Query Engine provided the ability to query a triple store repository with a text box interface, while its Real-time feedback allowed users to construct valid queries. This semantic work is under continuous development, aiming to attribute new information retrieval opportunities for biology domain and other life sciences. Similar reasoning capabilities for interior design domain have been previously presented in [22], where a room space was annotated based on various parameters that come from a questionnaire. Its answers were broadcasted to the server in the form of HTTP requests that ultimately were synthesized into a single SPARQL query. The results of this query were then sent

back to the user as the potential decoration solutions that match their needs. The execution of queries was feasible thanks to the adoption of the SPARQL API which was served by the Jena Model interface. The last one was responsible for performing queries directly upon the OWL knowledge base, allowing in that way the retrieval of any information from OWL-DL ontologies. Consecutively, Jena was further used to reason with the existing ontologies of the implemented semantical system.

3.5 DECoration Ontology (DECO)

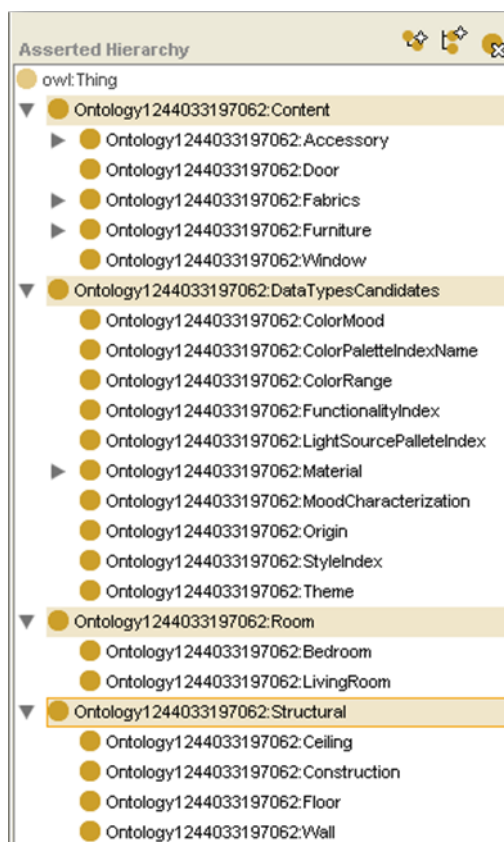
Our Semantic Web application is based on an interior design and decoration ontology that has been presented in [23]. DECO is an ontology implemented in OWL-DL language which deals with a single room space each time. Every room space is treated as a separate ontology consisted of the physical objects of this space and the annotation of the qualitative and quantitative characteristics of these objects and their corresponding space. Also, it's noteworthy that the architectural design of this ontology allows the annotation of products coming from any kind of external source, like decoration books, interior design photos or even from a storefront shop.

For the needs of our application we narrowed down the possible room spaces to the two most widely used interior spaces of a common residence which are "Bedroom" and "Living Room". DECO was slightly modified in order to keep up with the requirements stemming from these two spaces. Firstly, it was crucial to become a classification rearrangement for the sufficient support of these two room spaces, as well as the addendum of additional physical objects in the form of OWL classes, such as Carpets, Placemats, Sculptures, Curtains, etc. Secondly, there was a significant increment in the number of properties in order to properly define the characteristics of the newly added physical objects. The augmentation of DECO ontology with these properties ensures the smooth and optimal association between physical objects, the room space, and the intersection of these two.

DECO ontology defines a number of distinct classes beneath owl:Thing, the root of every OWL ontology. These classes have been implemented under careful research on the field of interior design and decoration, in order to fulfill the needs met by the room spaces where DECO ontology has to deal with. For that reason, it was deemed necessary the definition of four major classes as mentioned in paper [13]

- “Content”,
- “DataTypesCandidates”,
- “Room” and
- “Structural”

Each one of these classes contains subclasses for the annotation of a room space with physical objects and qualitative or quantitative characteristics. Each class and subclass of DECO ontology represents these real world concepts, which are used for the interior design and decoration of a room space. Though these core concepts of the domain have been already implemented in DECO ontology, there is always the capability to further expand or modify the existing concepts in the near future.



The subclass "Content" contains all physical objects of a room space such as windows, doors, furniture, accessories and others. These objects are subclasses which describe groups to related objects. The subclass "DataTypesCandidates" contains the qualitative characteristics of an interior space. Each one of these enumerators contains individuals which used for the assignment of characteristics. The subclass "Room" is a class that contains the interior spaces like living room and bedroom available for annotation. The subclass "Structural" is a class that contains subclasses such as floor, walls, ceiling and other objects which are core objects of the interior space that are built-in and not movable.

Figure 3-2. DECO Classes

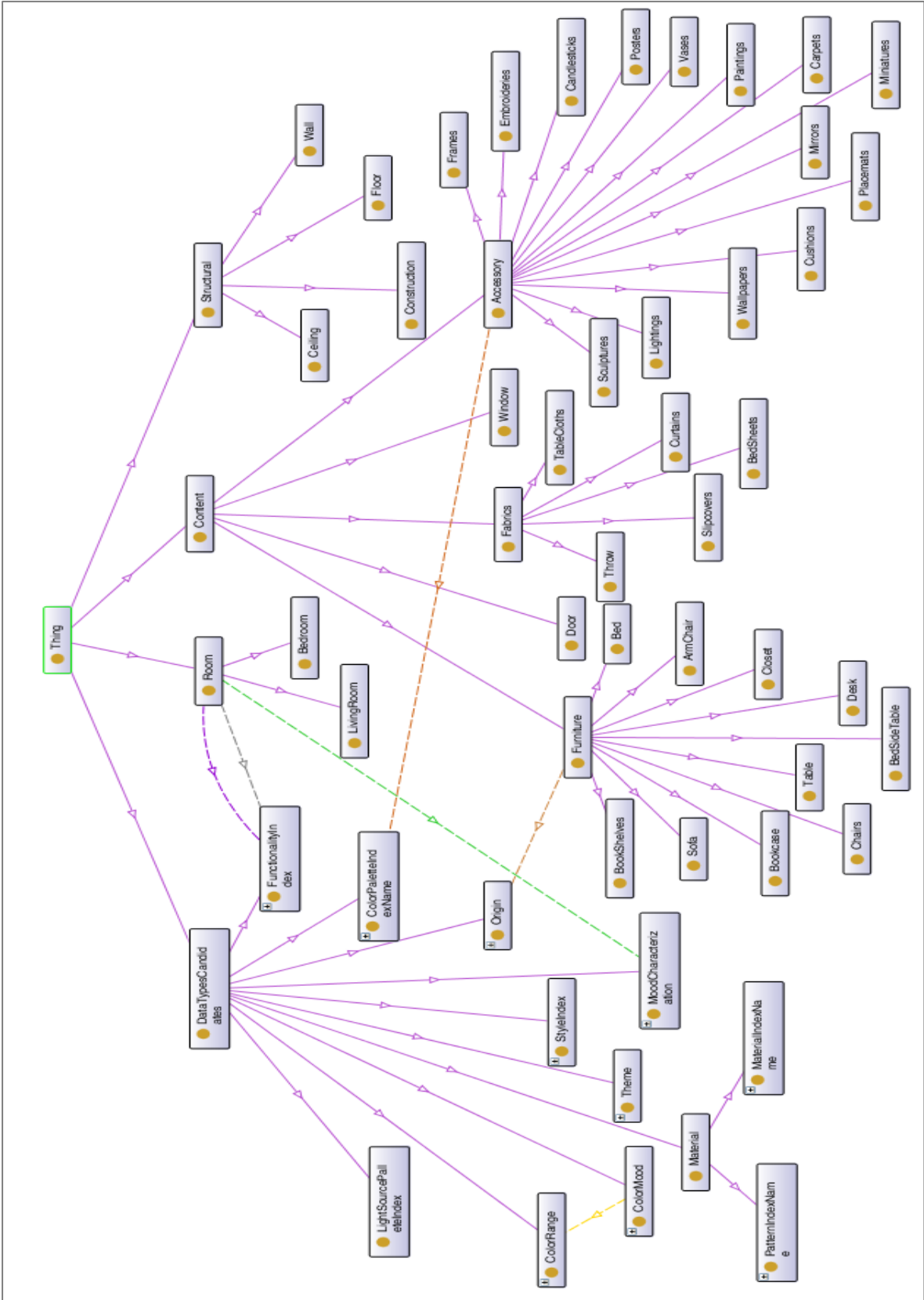


Figure 3-3. DECO ontology – graphical layout

DECO object properties aim to give the ontology fundamental relational functionalities by interconnecting classes in individuals. Object properties category is the first and most commonly used of the three major property categories provided by OWL. Any object property is defined as an instance of the built-in OWL class owl:ObjectProperty. Object properties link individuals from the domain area to individuals from the range area, which are used as axioms in reasoning. After having described the physical objects and the properties that are necessary in order to characterise them and associate them with other physical objects, we must define the domain and range areas that the properties can take values from. The rules for constructing OWL sentences concur to natural languages syntax where “Domain” reflects the “Subject”, the Property reflects the Verb and the “Range” reflects the object.

In specific:

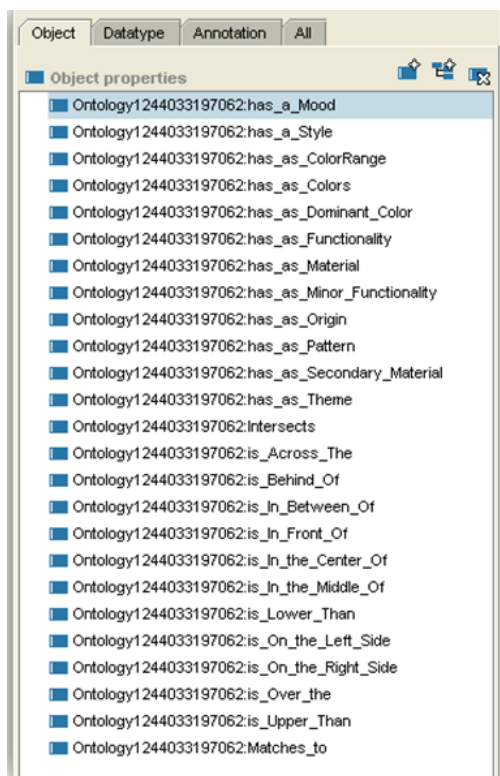


Figure 3-4. DECO object properties

- ❖ ***has_a_Mood***: property which defines that one individual of class “Room”, has personality which is the individual of class “MoodCharacterization”.
- ❖ ***has_a_Style***: property which define that one individual of class “Structural” or “Content” has style which is individual of class “StyleIndex”
- ❖ ***has_as_ColorRange***: property which defines that one individual of class “ColorMood” contains some general category, which is individual of class ColorRange.
- ❖ ***has_as_Colors***: property which defines that one individual of class “ColorRange” or “Structural” or “Content” has some color, which is individual of class “ColorPaletteIndexName”.
- ❖ ***has_as_Dominant_Color***: property which defines that one individual of class “Accessory” has a dominant color which is individual of class “ColorPaletteIndexName”.
- ❖ ***has_as_Functionality***: property which defines that one individual of class “Room” has basic (major) functionality which is individual of class “FuctionalityIndex”.

- ❖ ***has_as_Material:*** property which defines that one individual of class “Structural” or “Content” comes from some basic material which is individual of class “MaterialIndexName”.
- ❖ ***has_as_Minor_Functionality:*** property which defines that one individual of class “Room” has a secondary (minor) functionality which is individual of class “FuctionalityIndex”.
- ❖ ***has_as_Origin:*** property which defines that one individual of class “Furniture” has origin which is individual of class “Origin”.
- ❖ ***has_as_Pattern:*** property which defines that one individual of class “Structural” or “Content”, has a pattern which is individual of class “PatternIndexName”.
- ❖ ***has_as_Secondary_Material:*** property which defines that one individual of class “Structural” or “Content”, has a secondary material which is individual of class “MaterialIndexName”.
- ❖ ***has_as_Theme:*** property which defines that one individual of class “Posters”, ”Paintings” or “Wallpapers” has a theme which is individual of class “Theme”.
- ❖ ***Intersects:*** property which defines that one individual of class “Structural” or “Content” intersects with some other individual of class “Structural” or “Content”.
- ❖ ***is_Across_The:*** property which defines that one individual of class “Structural” or “Content” is across of other individual of class “Structural” or “Content”.
- ❖ ***is_Behind_Of:*** property which defines that one individual of class “Structural” or “Content” is behind of some other individual of class “Structural” or “Content”.
- ❖ ***is_In_Between_Of:*** property which defines that one individual of class “Structural” or “Content” is in between of some other individual of class “Structural” or “Content”.
- ❖ ***is_In_Front_Of:*** property which defines that one individual of class “Structural” or “Content” is in front of some other individual of class “Structural” or “Content”.
- ❖ ***is_In_the_Center_Of:*** property which defines that one individual of class “Structural” or “Content” is in the center of some other individual of class “Structural” or “Content”.
- ❖ ***is_In_the_Middle_Of:*** property which defines that one individual of class “Structural” or “Content” is in the middle of some other individual of class “Structural” or “Content”.
- ❖ ***is_Lower_Than:*** property which defines that one individual of class “Structural” or “Content” is lower than other individual of class “Structural” or “Content”.
- ❖ ***is_On_the_Left_Side:*** property which defines that one individual of class “Structural” or “Content” is on the left side of some other individual of class “Structural” or “Content”.

- ❖ *is_On_the_Right_Side*: property which defines that one individual of class “Structural” or “Content” is on the right side of some other individual of class “Structural” or “Content”.
- ❖ *is_Over_the*: property which defines that one individual of class “Structural” or “Content” is over of some other individual of class “Structural” or “Content”.
- ❖ *is_Upper_Than*: property which defines that one individual of class “Structural” or “Content” is upper than other individual of class “Structural” or “Content”.
- ❖ *Matches_to*: property which defines that one individual of class “Structural” or “Content”, matches to some other individual of class “Structural” or “Content”.

DECO datatype properties category is the second of the three major property categories provided by OWL. Any datatype property is defined as an instance of the built-in OWL class owl:DatatypeProperty. Unlike object properties where they form relationships between individuals, datatype properties are used to link OWL individuals with typical data values. This category of properties is quite flexible, since they have the capability to be transformed into either object or annotation properties. The counterpart object property is feasible with the addition of an OWL class relevant to the range information of the datatype property. Similarly, the counterpart annotation property is identical to its datatype property, but comes at the cost of reasoning support. Due to the nature of annotation properties, it is not possible the deduction of knowledge by a reasoner which may has been attached to an ontology.

High significance is the ”has a 3D Model” property of an individual of class ”Structural”, ”Content” and ”Room”, which is a string containing the URI of an X3D model of this individual. This is important for the visualization interface of the platform, where these models are to be drawn from a local or network repository and composited into a VR representation of the room.

In specific:

- ❖ *has_a_3D_Model*: property which defines that one individual of class “Structural”, “Content” or “Room” has a string which is URI of a 3D model of this individual.
- ❖ *has_a_Covering_Factor*: property which defines that one individual of class “Room” has a float number which is represent the percentage of room that covered by objects.
- ❖ *has_a_Approximate_Size*: property which defines that one individual of class “Room” has a string which represents the approximate size of the room.



Figure 3-5. DECO datatype properties

- ❖ ***has_as_ColorPalletteIndexNumber:*** property which defines that one individual of class “ColorPalletteIndexName”, has a string which defines a unique number that represents the color.
- ❖ ***has_as_ColorPalletteIndexURL:*** property which defines that one individual of class “ColorPalletteIndexName” has a string which indicates the URL for the availability of color.
- ❖ ***has_as_Function_Type:*** property which defines that one individual of class “Curtains” has a string which represents the functionality of curtains. This string has functional values which are: Simple, Blinds and Roman.
- ❖ ***has_as_Functional_Height_Dimension:*** property which defines that one individual of class “Door”, “Furniture”, “Window” has a float number which represents the height that the object is functional.
- ❖ ***has_as_Functional_Length_Dimension:*** property which defines that one individual of class “Door”, “Furniture”, “Window” has a float number which represents the length that the object is functional.
- ❖ ***has_as_Functional_Width_Dimension:*** property which defines that one individual of class “Door”, “Furniture”, “Window” has a float number which represents the width that the object is functional.
- ❖ ***has_as_Gender_Type:*** property which defines that one individual of class “Furniture” has a string which represents the sex group of furniture. This string has functional values which are Male and Female.
- ❖ ***has_as_Height_Dimension:*** property which defines that one individual of class “Structural” or “Content” has a float number which corresponds to the height dimension of the object.
- ❖ ***has_as_Hexadecimal:*** property which defines that one individual of class “ColorPalletteIndexName” has a string which defines the hexadecimal number of color.

- ❖ ***has_as_Illumination_Target***: property which defines that one individual of class “Lightings” has a string which represents the type of lightings. This string has functional values which are: Spotlight, Pointlight and Directional.
- ❖ ***has_as_Length_Dimension***: property which defines that one individual of class “Structural” or “Content” has a float number which corresponds to the length dimension of the object.
- ❖ ***has_as_Luminosity***: property which defines that one individual of class “Room” has a float number which represents the luminosity of the interior space.
- ❖ ***has_as_MaterialIndexURL***: property which defines that one individual of class “MaterialIndexName” has a string which indicates the URL for the availability of the material.
- ❖ ***has_as_PatternIndexURL***: property which defines that one individual of class “PatternIndexName” has a string which indicates the URL for the availability of the pattern.
- ❖ ***has_as_Placement_Target***: property which defines that one individual of class “Lightings” has a string which corresponds to the placement of object in the room. This string has functional values which are Ceiling, Freestanding, Desktop and Wall.
- ❖ ***has_as_Residence_Location***: property which defines that one individual of class “Room” has a string which corresponds to the location of the room. This string has functional values which are: Mountain, Seaside and Urban.
- ❖ ***has_as_Residence_Type***: property which defines that one individual of class “Room” has a string which corresponds in the residence type. This string has functional values which are Main and Secondary.
- ❖ ***has_as_Shape***: property which defines that one individual of class “Structural” or “Content” has a string which describes the characteristics of a physical object.
- ❖ ***has_as_Size***: property which defines that one individual of class “Accessory” has a string which corresponds to the appropriate size of an object.
- ❖ ***has_as_TargetGroup_for_BedRoom***: property which defines that one individual of class “Bedroom” has a string which corresponds to the target groups of bedroom. This string has functional values which are “Family” and “Student”.
- ❖ ***has_as_TargetGroup_for_LivingRoom***: property which defines that one individual of class “LivingRoom” has a string which corresponds to the target group of living room. This string has functional values which are “Family” and “Student”.

- ❖ ***has_as_Technotropia***: property which defines that one individual of class “Fabrics” has a string which sets the style that is built each fabric.
- ❖ ***has_as_TopViewURL***: property which defines that one individual of class “Room” has a string which indicates the URL or simply contains information about the floor plan of the room.
- ❖ ***has_as_Width_Dimension***: property which defines that one individual of class “Structural” or “Content” has a float number which corresponds to width dimension of object.
- ❖ ***has_DoubleBed***: property which defines that one individual of class “Bed” has a string which corresponds to double bed. This string has functional values which are: Yes and No.
- ❖ ***made_From***: property which defines that one individual of class “Accessory” or “Furniture” has a string which corresponds to the method of construction of the object. This string has functional values which are: Industrial and Handmade.
- ❖ ***number_of_Females***: property which defines that one individual of class “Bedroom” has an integer (int) number which corresponds to the number of females which will use the bedroom.
- ❖ ***number_of_Males***: property which defines that one individual of class “Bedroom” has an integer (int) number which corresponds to the number of males which will use the bedroom.
- ❖ ***to_be_Placed_at***: property which defines that one individual of class “Curtains” has a string which corresponds to the curtains’ placement. This string has functional values which are: Windows and Door.

3.6 DECO Web-based Ontology Editor

The application was developed with Jena framework and takes advantage of its various integrated APIs for the manipulation of DECO ontology. The whole process of room annotation is exclusively done from Jena framework, while the presentation and user interaction is assisted by various technologies like Java Servlets, JSP, JavaScript and CSS. The front-end of the application is implemented in HTML, where the users are navigated through a series of forms, in order to efficiently annotate a room space, or they are requested to fill a questionnaire and retrieve possible matches from a database of room spaces. The concepts used for the room space annotation and decoration are based on the DECO ontology presented in Subchapter 3.5, which has been totally fused into the application. However, in order to prevent the arbitrary input of characteristics to a room space - which could result in a vast number of rooms with meaningless or unsuitable

characteristics - all possible options of these characteristics are derived from a prototype ontology named "master ontology". This ontology is the basis for the creation of every new room space and possess all the available qualitative and quantitative characteristics a room space may have, which can be enriched or removed anytime by the appropriate user.

Our semantic web interface has mechanisms to store a room space with two different formats.

- The file-backed storage, as a dedicated RDF/XML file on the disk, can be accessed anytime in order to view the current characteristics of this room space, or change them according to the latest decoration ideas, fashion trends, or design aspects
- The RDBMS-backed storage, where Model datasets are persisted in the form of statements on a MySQL database. This storage type is taking place through the collaboration of the Jena SDB triple store system and the underlying operations provided by the MySQL database

These mechanisms provide easy and fast way to store a room space, as an OWL ontology. Our wizard supports two kind of users Content Providers and Designers for this procedure.

- **Content Providers:** The first step in the process is the population of the knowledge base with items to be used in rooms. A content provider, such as a furniture or textile retailer can use the web-based platform interface to assign individuals to classes and define their properties. Using the Web interface, a furniture retail company can introduce their chosen items into the ontology, and specify their characteristics. The object materials, colors and design style can be selected from the ontology, and X3D models of the objects can be provided by them for incorporation into the knowledge base.
- **Designers:** Using the same wizard to access the knowledge base, entire new rooms can be created. Using essentially the very same approach, a designer accesses the web wizard interface and organizes a fundamental room structure.

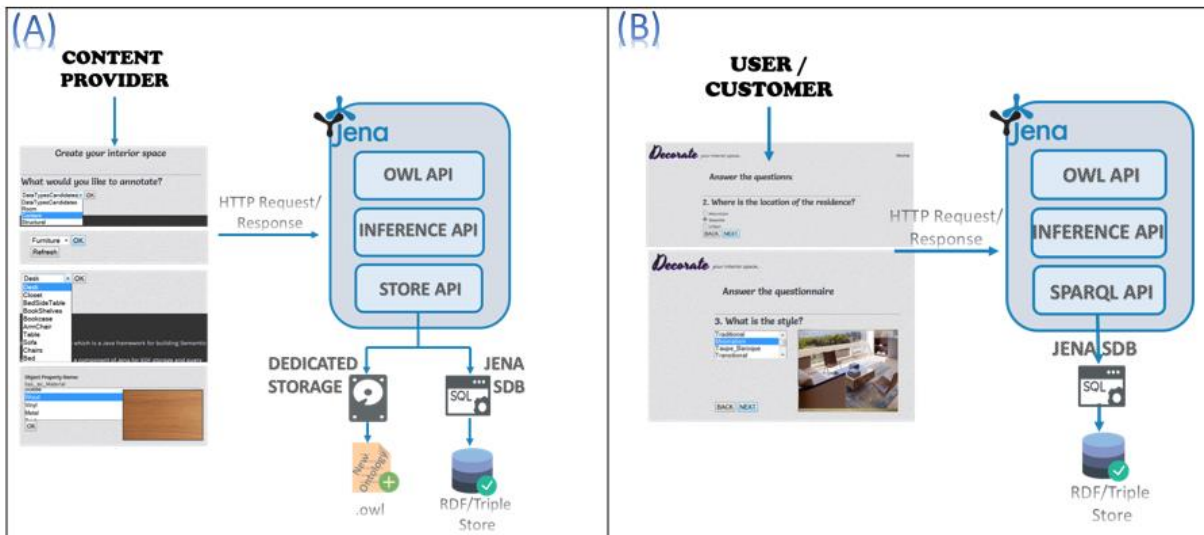


Figure 3-6. DECO architecture (A) Content Provider, (B) SPARQL mechanism

On the other hand, consumers were able to query and search all the ontologies through an easy-to-answer questionnaire backed by an SDB Triple Store mechanism (see Figure 3-6B). Reasoning capabilities for interior design domain have been previously presented in [24], where a room space was annotated based on various parameters that come from a questionnaire. Its answers were broadcasted to the server in the form of HTTP requests that ultimately were synthesized into a single SPARQL query. The results of this query were then sent back to the user as the potential decoration solutions that match their needs. The system takes advantage of SPARQL to query the ontology for instances matching particular wishes or needs.

Chapter 4 - Web3D

4.1 X3D Standard

X3D standard comes with a long history of representing web-based 3D graphics, since it constitutes the successor of the legacy VRML (Virtual Reality Modeling Language) [25]. It makes use of the XML format to encode its models, while the latest standard has been further enriched with advanced capabilities, like physical simulation and authentication services. Nowadays, X3D is the most widely used standard for the presentation of 3D content on the Web, using efficient XML validation mechanisms for the connotation of scripting errors to content's authors. Its versatile architecture supports and collaborates with various ISO standards, achieving in that way steady results in numerous combinations of hardware and software environments. Moreover, X3D specification defines several profiles with each one of them describing functionalities for distinct target groups. To be more specific, each one grants a transitional level of features' support, starting from advanced implementations on personal computers, down to basic functionalities for lightweight devices like tablets and smartphones. These profiles are shortly described below, followed by Figure 4-1 which presents X3D standard's architecture:

- **Core** profile designed towards minimal scenes and it is not used for ordinary works, since it contains only the absolutely necessary X3D nodes. For this reason, it is seldom meet in any kind of applications.
- **Interchange** is the main profile of X3D which is designed to support various features for use in 3D applications, like basic geometry, appearance and animation. This profile is also quite easy to be integrated in applications, due to the fact that there is not any runtime rendering model.
- **Interactive** profile is an enhanced version of the Interchange profile, enriched with navigation and interaction capabilities. Components of an X3D scene can now reference an external model via the Inline node, and make use of sensor (e.g. TouchSensor, PlaneSensor, etc.) or lighting (e.g. PointLight, Spotlight) nodes.
- **Immersive** profile extends Interactive profile by enabling 3D graphics that include audio, collision, scripting, and fog nodes. These features maximize standard's usage in most gaming applications, making this profile to continuously gain ground compared to the rest ones.

- **Full** profile includes the entire set of nodes that are defined in X3D specification. This profile contains very specific usage components like Non-Uniform Rational B-spline Surfaces (NURBS), Distributed Interactive Simulation (DIS), GeoSpatial, and Humanoid Animation (H-Anim).

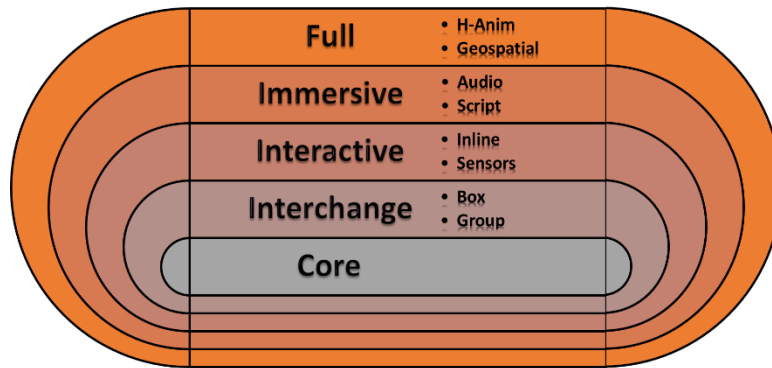


Figure 4-1. X3D Architecture

Such a profile-tiered architecture was initially proposed by 3D community as the mean to deliver a multipurpose balanced system for web-based applications. Taken into consideration their proposal, Web3D Consortium introduced the aforementioned sequence with a couple of additional X3D profiles, the **MPEG-4 Interactive**, **CDF (CAD Distillation Format)** and **Medical-Interchange**. Last but not least, authors can anytime extend X3D standard using a native prototyping mechanism that allows them to create new nodes or modify the already defined. Today, X3D standard has been adopted and become the ideal representation type for the majority of XML-based languages like HTML5 and XHTML. Its agile file format has enabled the implementation of 3D virtual representations in various domains like architecture, multimedia, medical, and much more.

4.2 X3DOM

Despite the fact that X3D has been extensively used in HTML5 in order to represent 3D content, its specification does not address the problem of connectivity between the user's browser (frontend) and a X3D model (backend). However, a proposal to overcome this issue through an architecture named X3DOM (pronounced X-Freedom) was raised in [26], incorporating the technologies of X3D, WebGL, HTML, CSS and JavaScript under a common framework. The

presented model was soon made available to the public as an open-source JavaScript library capable of rendering X3D content on the client side. Its functionality was based on the manipulation of X3D elements as active parts of the HTML5 DOM tree without using plugins.

The latest works in this area have focused on the development of 3D virtual environments with the use of X3DOM. Their main scope was to increase reality level in Web3D scenes through the enhancement of an X3DOM structure. For this reason, many authors proposed the extension of the X3DOM specification sheet with special nodes for each occasion. One of the most remarkable attempts introduced a set of spatial sound components made exclusively for the X3DOM framework [27], while other implementations enriched X3DOM with rigid body physics [28] or presented HD video streaming mechanisms in 3D Virtual Reality environments [29]. Finally, the last year was also proposed the upgrade of shadow representation to X3DOM framework changing the tide of interest from geometry and networking, to visual rendering techniques [30].

Summarizing, we can safely deduce that X3DOM is able to incorporate 3D content on the Web without the installation of plugins, while at the same time, it provides an efficient integration mechanism for the current Web standards. Furthermore, its novelty lies to the merging of HTML5 and X3D standards through a viable framework, which is supported by the majority of today's browsers.

4.3 Build an X3D file

We have the ability to build an X3D scene with many programs, like 3D Studio Max, Vivaty Studio, Blender Suite and others. Each one of these programs, enable many functionalities for the modification of the X3D elements. For example, we create a box with Vivaty Studio and we changed the size and the color of the element. The saving of this scene produces an .x3d file with the following code.

<pre><?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.0//EN" "http://www.web3d.org/specifications/x3d-3.0.dtd"> <X3D profile='Immersive' > <head> <meta name='ExportTime' content='12:30:12'/> <meta name='ExportDate' content='8/30/2016'/> </head></pre>	X3D Code
--	----------

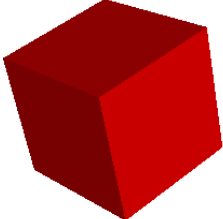
<pre> <Scene> <Shape DEF='Box1'> <Appearance containerField='appearance'> <Material DEF='Red' containerField='material' diffuseColor='1 0 0' /> </Appearance> <Box DEF='GeoBox1' containerField='geometry' size='5 5 5' /> </Shape> </Scene> </X3D> </pre>	<p>X3D Code</p>
	<p>X3D representation</p>

Table 4-1. The creation of a “box” with X3D

As we can see, the code is simple and we can write the x3d format to a text editor. The programs that are mentioned are responsible for the creation of more complex elements and 3D scenes.



Figure 4-2. Examples of X3D visual representations

The X3D has the following structure.

- File header
- X3D header
- Profile statement

- Component statement
- Meta tag statement
- X3D root
- X3D Scene

The `<X3D>` node contains all the tags for the representation of an X3D scene. As a result, the whole 3D content is described inside of the `<scene>` tag. A scene can have many nodes like shape, group, lights, viewpoint etc.

4.4 Setup and run X3DOM

All you need for setting up and running an X3DOM example is one text-editor and a web browser. At first, we create a new file and we add the X3DOM JavaScript file and CSS file to the header of this HTML file. Then, we are ready to add some 3D content to our page. We copy the same code from the previous example and we paste the same code within html page and x3d tag. Then, we add another box with blue color but with different transform values. The code for the x3d scene is the following:

```
<html>
  <head>
    <title>My first X3DOM </title>
    <script type='text/javascript' src='http://www.x3dom.org/download/x3dom.js'>
    </script>
    <link rel='stylesheet' type='text/css' href='http://www.x3dom.org/download/x3dom.css'>
    </link>
  </head>
  <body>
    <h1>My X3DOM Scene!</h1>
    <p>
      <x3d width='600px' height='400px'>
        <scene>
          <Shape DEF='RedBox'>
            <Appearance containerField='appearance'>
              <Material DEF='Red' containerField='material'
                diffuseColor='1 0 0'>
            </Appearance>
            <Box DEF='GeoBox1' containerField='geometry' size='5 5
5' />
          </Shape>
          <transform translation='6 5 0'>
            <Shape DEF='BlueBox'>
              <Appearance containerField='appearance'>
```

```

        <Material DEF='Blue' containerField='material'
            diffuseColor='0 0 1' />
        </Appearance>
        <Box DEF='GeoBox2' containerField='geometry'
            size='3 3 3' />
    </Shape>
</transform>
</scene>
</x3d>
</p>
</body>
</html>

```

Table 4-2. Code for inserting X3D in HTML page

The previous code is rendered from a web browser with the following format. Figure 4-3 shows the code in the inspector. As we observe the x3dom and the browser adds extra fields in the x3dom code.

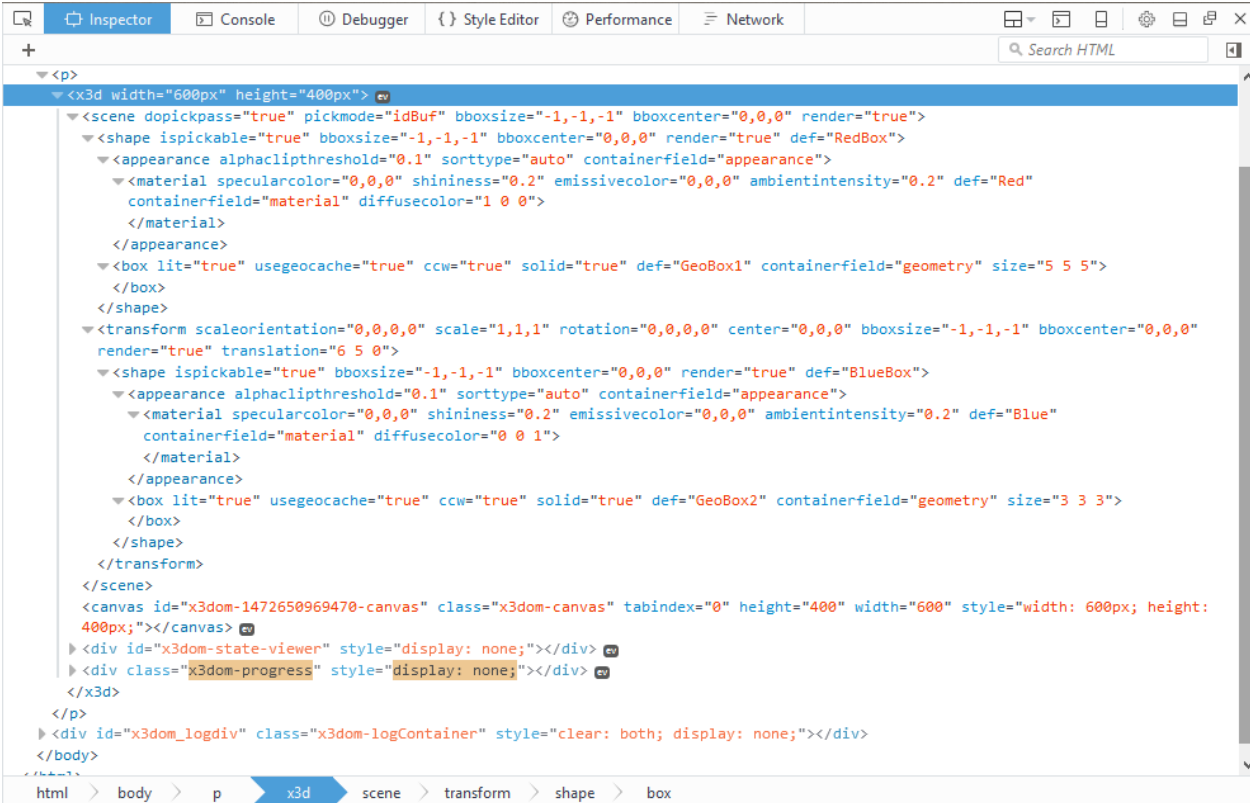


Figure 4-3. X3D rendering in browser

The images below demonstrate how to take extra information from X3DOM. When a web page with X3DOM has loaded, we press the hotkey “d” from our keyboard. With this way, we open the log window which shows information about our scene.

Figure 4-4 shows information about the object we click with the mouse and the coordinates of this position.



Figure 4-4. X3DOM debug mode for position coordinates

When we press “v” from our keyboard, we have information for the viewpoint in the log window, as shown in Figure 4-5.

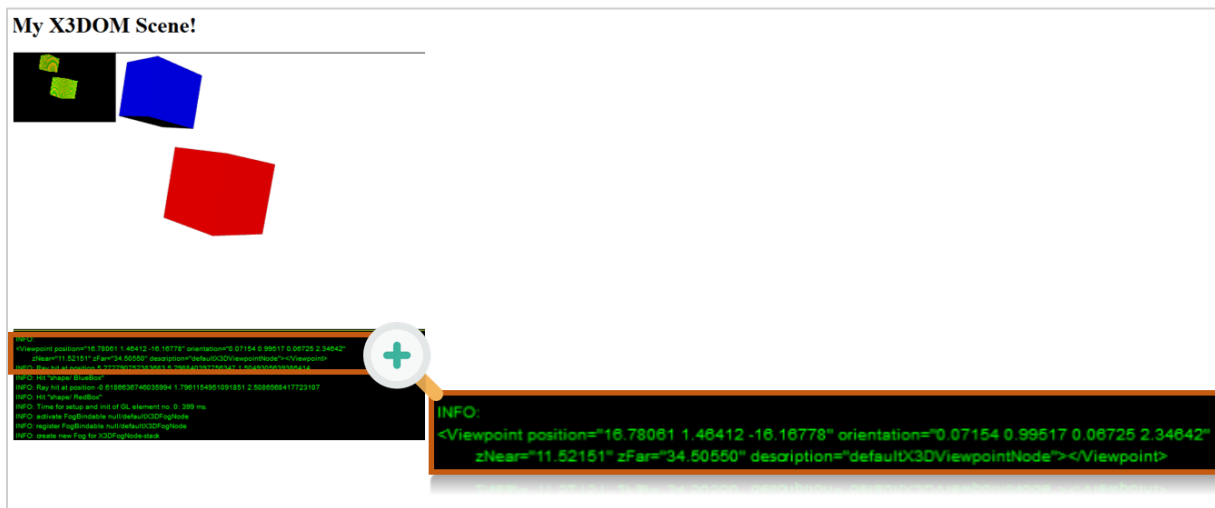


Figure 4-5. X3DOM debug mode for viewpoint

Sometimes, we want to include an external X3D file to our web page with X3DOM framework. X3DOM gives this ability to the users with the Inline node. By this node is able to add the x3d file in the x3dom structure. For the manipulation of this model, we can use the true value of mapDEFToID attribute.

```
<x3d width='500px' height='400px'>
  <scene>
    <inline namespaceName="Example" mapDEFToID="true" url="example.x3d">
    </inline>
  </scene>
</x3d>
```

Table 4-3. Add an external file to X3DOM scene

This attribute enables the access of the specified model by its ID value. We have access to every node of this model by using the namespace prefix and double underscore, like

```
document.getElementById('Example__part1');
```

4.5 Related Work

4.5.1 X3D Visualization solutions

A three-dimensional visualization of a product is definitely a crucial advantage for the sales of any e-commerce application compared to its plain image or 2D representation, since the user is able to imagine it inside its own room space. This fact was further strengthened by the rapid growth of various Web3D technologies that provide the customer with the ability to get involved in product's design process. So, today users can freely manipulate and customize the products they are interested in, through convenient and straightforward 3D visualization tools that emulate real world actions [31] These new experiences enabled by Web3D, didn't change the face of the classic World Wide Web, remained independent of the underlying platform, and their only requirement was the partial use of a browser plugin in a typical computer [32]. In the following paragraph are briefly described the most remarkable implementations on the Web, which are based on interior design concepts and employ Web3D solutions for the presentation and semantics of their datasets.

Lapeyre was one of the first web-based applications for the live preview of 2D objects on a same dimensionality environment, making use of the Away3D engine accompanied by Flash

platform. Its main advantage was the simplicity of the provided interface and the capability to dynamically change the color and size of objects (e.g. doors, gate, etc.). However, other solutions raised up for the representation of 3D objects in a 2D environment, by building a collaboration network of Java and OpenGL programming languages. In this way, it was made feasible the management of the depth factor of any three-dimensional model to its corresponding two-dimensional environment. Such a 3D solution was presented at “Homestyler”, an online application that allows users to configure a home on the Web. Application’s visitors start from an empty environment and a simple plan, and go through adding various objects like walls, windows, doors, furniture, etc. Even though the original configuration is based on a 2D environment, the final result is displayed under a 3D point of view.

Moreover, recent works [33] showed that Web3D applications keep gaining ground compared to their 2D or pseudo-3D alternative solutions. One of them was “Caidou”, a Web3D application that allowed customers to configure and view their personal portal on the Web. Such customers had the opportunity to visualize a 3D artificial representation of a 2D object or its imported photo, while the outcome could be anytime exported to a PDF document in PDF3D format. Following the same path, other solutions made use of Unity3D software to create web-based applications for 3D domain. At first, user selects the 3D model he desires, then he positions it in front of an image that corresponds to a predefined environment, drag and drop it in that environment –and finally- change its color to match with the presented scene.

In paper [34] was presented an architectural design mechanism for the automated conversion of SVG indoor plans to their corresponding X3D representations, taking into consideration the directional and scaling variables of these scenes and their contents. A similar approach was also adopted in [24] , where the design of the room space was taking place through a convenient SVG environment, backed by an ontological editor and an XSLT transformation algorithm.

4.6 DECO Web Editor

Our 3D viewer -namely “Deco Web Editor”- is based on X3DOM framework, ensuring in this way its unimpeded operation without the need of any plugin or pre-installed software. The implemented viewer is able to illustrate a room space which can be further manipulated via various X3DOM features. Such features have been integrated into the GUI of the web editor and include

the most widely known operations met on typical 2D/3D representation environments and the “Moveable” functionality from X3DOM framework.

The authored code makes use of X3DOM’s core functions, allowing us to apply changes upon an X3D scene with advanced manipulation and great flexibility capabilities. In the example below, we present an easy way to automatically add an event listener for mouse clicks to all "Transform" nodes of our 3D representation, enabling also the “Moveable” feature to these nodes.

```
var lastSelectedObject = null;
handleClick = function (element) {
    lastSelectedObject = element.hitObject;
    new x3dom.Moveable(boxes, lastSelectedObject.parentNode, moveCallback,
0);
};
main = function main() {
    window.requestAnimationFrame(main);
    x = document.getElementsByTagName("Inline");
    if (window.document.readyState === "complete") {
        for (var i = 0; i < x.length; i++) {
            if (x[i]._x3domNode._cf.children.nodes[0]) {
                for (var e in x[i]._x3domNode._cf.children.nodes[0]._childNodes) {
                    if(x3dom.isa(x[i]._x3domNode._cf.children.nodes[0]._childNodes[e],
x3dom.nodeTypeTypes.Transform)) {
                        if (x3dom.isa(x[i]._x3domNode._cf.children.nodes[0].
_childNodes[e], x3dom.nodeTypeTypes.Transform)) {
                            x[i]._x3domNode._cf.children.nodes[0]._childNodes[e]
._xmlNode.addEventListener("click", handleClick,
false);
                        }
                    }
                }
            }
        }
    }
}
```

Table 4-4. Adding listener to X3DOM nodes

As a result, when a user clicks to a furniture or a piece of furniture, he has the ability to "drag" the furniture at any point he wants. Additionally, the manipulation of the physical state of a 3D object is feasible via the provision of a web-based editor that comes in the form of a menu item. The code in Table 4-5 displays how to change the color of a 3D object. When the JavaScript code takes as input the new color from the “value” field of the color-picker, an iteration structure is run to find the “Appearance” node of this specific object. After this, the same part of code updates the value of the “diffuseColor” attribute and the color is successfully changed.

```

var color = document.getElementById("form_color");
color.onchange = function changeColor()
{
    var newCol = document.getElementById("form_color").value;
    for (child in lastSelectedObject.childNodes) {
        if (x3dom.isa(lastSelectedObject.childNodes[child]._x3domNode,
            x3dom.nodeTypes.Appearance)) {

lastSelectedObject.childNodes[child]._x3domNode._cf.material.node
    ._xmlNode.setAttribute('diffuseColor', newCol);
        x3dom.reload();
        socket.emit('changeColor', {'type': 'changeColor', 'object':
lastSelectedObject.getAttribute("id"), 'color': newCol});
        }
    }
};

```

Table 4-5. JavaScript code for changing color in X3DOM node

DECO Web Editor enables the representation of any X3D scene in user's browser via X3DOM framework. For the demonstration purposes of this thesis we have created a set of living rooms with the assistance of Vivaty Studio, where a couple of them are shown in the following Figure.



Figure 4-6. A set of living rooms which have created for this thesis

Chapter 5 - Web Real-Time Communications (WebRTC)

One of the most significant steps forward in web browser connectivity since the launch of AJAX by Google back in 2004 is definitely the WebRTC project. In the meanwhile, the Asynchronous Javascript and XML (AJAX) approach enabled the updating of specific webpage's components without the need of full page reloads, and in 2008 a bi-directional communication protocol named WebSockets made its appearance on web browsers. Figure 5-1 displays the development path of communication protocols on the Web until WebRTC.

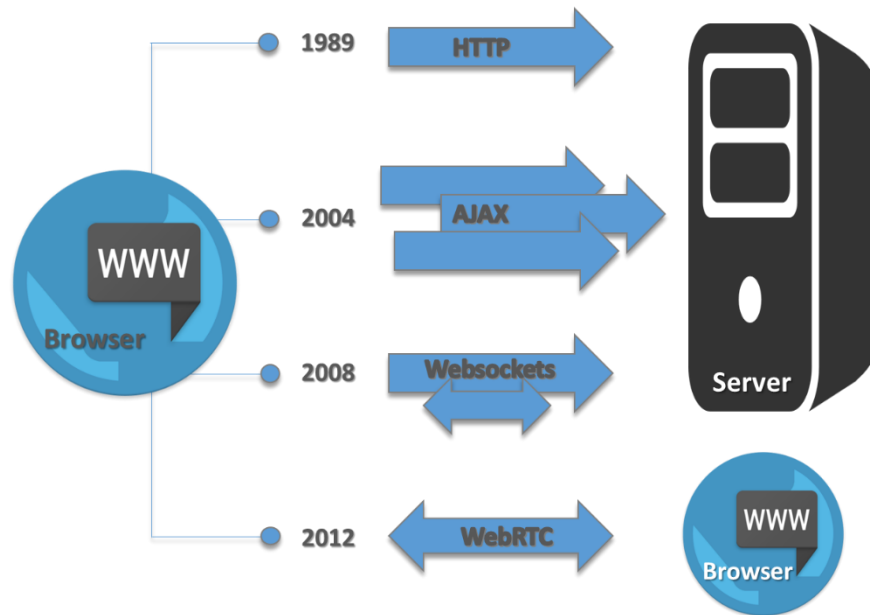


Figure 5-1. The history of communication protocols until WebRTC

The Web Real-Time Communications (WebRTC) standard increases the multimedia capabilities of today's browsers by providing an API for the direct real time communication between two or more computers. This interaction takes place in users' web browser without the use of extra plugins, making easy to share data and multimedia (video and audio). WebRTC is a standard which was developed by the World Wide Web Consortium (W3C) and the Internet Engineering Task Force (IETF) as a free, open source project, that aims not only to enrich current browsers, but also the mobile applications with the functionalities provided by the Real-Time Communications APIs. The vision of WebRTC is to merge the communication of diverse devices

(phone, TV, computer, etc.) under a common platform with just a few lines of JavaScript code. Figure 5-2 shows the possible participants in a WebRTC system, mixing browsers from different operation systems (PCs, tablets, phones) and additional elements like PSTN gateways and others.

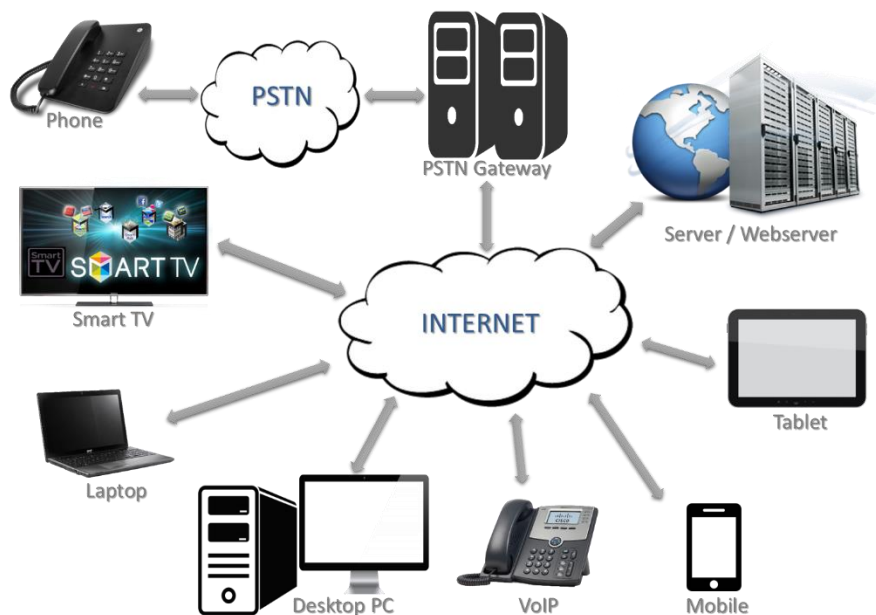


Figure 5-2. Multipurpose WebRTC system

WebRTC standard consists of three different APIs:

- The first API is called **getUserMedia** and provides a homonym method that exists in window.navigator object. It allows access to the multimedia streams coming from local devices -like camera and microphone- and displays them into user's browser. The implemented method comes with a constraints parameter and an object with audio and video properties. When the latter two properties are attributed with the "true" Boolean value, then the web browser is forced to request a video or audio streaming correspondingly. On the other hand, a "false" value denotes an idle state where nothing happens. The underlying API helps developers and users to access local devices with a single function without installing additional software or plugins to their web browsers. The getUserMedia API is currently available in Chrome, Firefox, Opera and Edge.
- The second API called **RTCPeerConnection** represents a WebRTC connection between the local computer and a remote peer. It is used to handle streaming of audio, video and

data between two participant peers. It is noteworthy that this API comes with different names depending on the browser, where Google Chrome and Opera browsers make use of the `webkitRTCPeerConnection` name, while Mozilla Firefox names it as `mozRTCPeerConnection`.

- The third API is called **RTCDataChannel** and makes feasible the transfer of data from one peer to another. It closely collaborates with `RTCPeerConnection` API and takes advantage of the Stream Control Transmission Protocol (SCTP) for connectivity purposes and the transmission of data.

5.1 WebRTC Network Protocols

Network Address Translation (NAT) gives the opportunity to the device in a private local network to get a public IP address. WebRTC cannot be used without a public IP address for the negotiation phase. To success NAT traversal WebRTC uses three different types of network protocols, which are STUN, TURN and ICE.

STUN

Session Traversal Utilities for NAT (STUN) is a client-server protocol. The server part discovers the user's IP address that is public and the TCP port. The STUN client be noticed for these. This give to the WebRTC peer the opportunity to get its public IP address and port, and passes them via WebRTC signaling to the other peer. STUN gives permission to the WebRTC media and data flows like real-time communication with voice, video, and messaging between peers.

TURN

Traversal Using Relays around NAT (TURN) is used as a fallback solution to STUN when STUN cannot be used, due to firewall system that is installed in the LANs. TURN server offers to the WebRTC peer the compulsory public IP address for communication outside its LAN.

ICE

Interactive Connectivity Establishment (ICE) is a framework for connecting peers. The mission of ICE is to find the best solution for WebRTC peers choosing between STUN and TURN. First time, ICE tries to connect peers directly via STUN and if this fails ICE uses a TURN server.

5.2 WebRTC Network Topologies

For WebRTC-based conferences, where more than two users/peers participate Mesh, Star and MCU topologies can be used.

- In **Mesh topology** each peer connects directly to all other available peers keeping several peer-to-peer connections. Mesh topology is easy to be established but it can be used for limited peers.
- In **Star topology** one peer has the role of coordinator and holds the connections with any other peer. The central peer having the obligation to send the multiplexed data to all the other peers. The disadvantage is that the central peer should be robust enough to resist this load, while the other peers haven't special requirements.
- The vigorous network topology for conference communications is the **MCU (Multipoint Control Unit or Multipoint Conference Unit)**. MCU is used as a bridge. It is a server which is used when big number of participants want to connect to a single session and distribute multiplexed media content to each other. Whenever a participant (peer) decides to leave the specified conference, the MCU simply terminates this connection. A MCU can handle different codecs and resolutions to enable the interoperability between the participants.

5.3 Related Work

5.3.1 WebRTC solutions

The HTML5 protocol and an extended version of MPEG-7 standard were used to form a modular cloud-based distributed platform for Virtual Reality Advertising [35] with improved scalability and content management features. The architecture of the system provides interactive banners which are served with different interfaces and manipulation mechanisms, according to the input device of its users (mobile, desktop, etc.). In this way, it is guaranteed a high responsiveness real-time experience and the intuitive control of small size screens.

A more recent attempt [36] to enable real-time communication over the Web introduced an innovative architecture with video and chat capabilities. Our work also provides a web-based implementation with video streaming and chat features without the need of any plugin installation. Moreover, a custom-made MCU makes feasible the conference-like communication between several peers. Last but not least, we plan to evolve the server logic of the entire application, in order to optimize and speed up the simultaneous communication of three or more clients.

The last few years various applications have been developed using WebRTC for a diversity of application domains, utilizing some of its several APIs for text messaging, video calling, conferencing, online gaming, etc. A recent example lies to ‘Firefox Hello’ add-on, a special feature that comes preinstalled with Mozilla Firefox browser [37]. It provides free video and audio calls via Firefox, Chrome and Opera browsers, without the need of external or additional plugins. Similar face-to-face video chat features are also included in OpenTok (TokBox), an application which is supported from the majority of website and mobile applications, as well as various “Web Call-Me” and “Click-to-Call” buttons provided by different developers.

Today, World Wide Web is able to provide Real-Time Communication (RTC) services between users’ browsers all around the Web thanks to WebRTC. The latter started as a simple JavaScript-based API aiming to attribute browsers and mobile applications with Real-Time Communication capabilities. In [38] was presented a middleware web-based platform that took advantage of various state-of-the-art technologies like HTML5, JavaScript, WebRTC and X3DOM. Their work made use of mobile devices’ sensors for statistical processing, in order to analyze and visualize the obtained data for future use in Internet of Things equipment.

With the passage of time, however, there were also applications that integrated WebRTC capabilities within three-dimensional virtual worlds for educational purposes. The majority of them dealt with the composition of a 3D collaborative environment for educational gaming that supported video calls, text messaging and the capability to select, insert, and manipulate 3D objects in an X3D scene using the Data Channel of WebRTC. Such a unification of Communication and Graphics fields is evident in [39], where X3DOM framework was merged with WebRTC technologies in order to constitute a virtual 3D collaborative environment for the cooperation of web peers at real time, while at the same time, these peers were able to manipulate the three-dimensional scene without the use of plugins. On the other hand, WebRTC was also found to be ideal for immersive video conference experiences, since recent demonstrations [40] showed that the implemented virtual worlds could be further enhanced with real-time social media connections, such as the Facebook, Twitter, YouTube, Spotify, and email as well.

5.4 Our WebRTC approach

While it's technically possible to make a point-to-point video call without anything other than a network between the two clients, it becomes difficult to create multi-party video calls without a hardware or software mean between the endpoints. This is where a Multipoint Conferencing Unit or Multi-Point Control Unit (MCU) comes in.

A MCU is a bridge that offers the ability to connect multiple users/participants to a single voice or video session. It makes use of a mixing architecture, where every participant sends its video or/and audio to a central server, while at the same time, it retrieves this server's mixed media stream of the other previously received participants. On the market, we meet many solutions for the WebRTC-based multi conferencing. Software like Intel Collaboration Suite for WebRTC [41], Kurento [42], Licode [43], Jitsi [44] are some of the media servers or APIs which are able to add real-time communication to our application.

For this thesis however, we have implemented our own MCU server for multi-party video call. Our approach not only enables peer-to-peer communication but also enables WebRTC-based video and audio conferencing.

Our WebRTC approach

- ❖ Is written in JavaScript Language
- ❖ Provides Groups / "Sessions" to the users to make multi conference calls

- ❖ Offers users access control and service registration roles
- ❖ Ensures data integrity through a secure channel of communication via HTTPS protocol,
- ❖ Supports Session Traversal Utilities for NAT (STUN server) which allows to an end host to expose its public IP address when it is located behind Network Address Translation (NAT) in various complicated conditions (e.g. with or without firewall). This ability enables the Interactive Connectivity Establishment (ICE) and helps devices to connect to each other under different networks.
- ❖ Uses the same signaling server (socket.io) under node.js for signaling messages, chat messages and video call messages. No other device or API was used for the creation of our MCU.
- ❖ All stream messages go through the MCU server, resulting to the reduction of the stream traffic
- ❖ Allows a wide range of devices (laptop with any operating system, smartphones, tablets with internet connection, etc.) to take advantage of our web application
- ❖ Has a responsive web design supporting multiple resolutions for the streaming of video to the client's webpage
- ❖ Enables Real Time Communication with 3D representation of a room space for all users who are in the same "session"
- ❖ Gives the ability to users to modify a 3D item and notifies other users to change this 3D item
- ❖ Provides Semantic Web solutions to the decoration areas according to users' desires

See the Subchapter 6.4.2 and Multi-Conferencing Module for more details.

Chapter 6 - Architecture and Prototype

The development of any collaborative environment is based on the establishment of a stable and secure communication protocol, implementing at the same time, all the necessary functionalities that its users may fulfil as subjects of the underlying domain. In the following subsections are described the X3DOM collaborative editor which developed with the use of WebRTC for Real Time Communication and Semantics for decoration and also the stages involved into a commonly used process flow from the perspective of an end-user. Our application provides an integrated framework for all types of users (content provider, designer, decorator, simple user) to communicate and directly exchange information.

6.1 Functional Design

On entering the application, user has the ability to log in with its credentials in order to “join” an available group and make use of the capabilities provided by the 3D collaborative environment. This environment not only supports the resizing, rescaling, relocation of physical objects and spaces, but it also comes with a semantic search of a desired object followed by its inclusion to the room space. Finally, the implemented application makes feasible the communication of an unrestricted number of users in each active group, using either video call or chat messaging services.

6.1.1 Login and Registration Manager

Upon entering application, user comes confront with the very first page of the system, which is none other than the Login/Registration page. The basic user roles for this system are decorators and users. The registered users using their credentials (username and password) have the choice to log in to the system and join a “session”, which is a decoration room with real-time communication abilities, Web3D functionalities and Semantic Web decoration solutions. In case they don’t have an account an account, the system displays a registration page. In this page, the user has to fill in some fields with personal information like his name, surname, email, username and password.

After the successful registration to our system, he is redirected to the login page for signing in and joining a “session” with other users.

The login form of our system is illustrated in Figure. 6-1A and requires the username and password of the user. The successful combination of these credentials or their potential mismatch are denoted by the alert messages displayed in Figure 6-1B, while Figure 6-1C depicts the fields of the registration form.

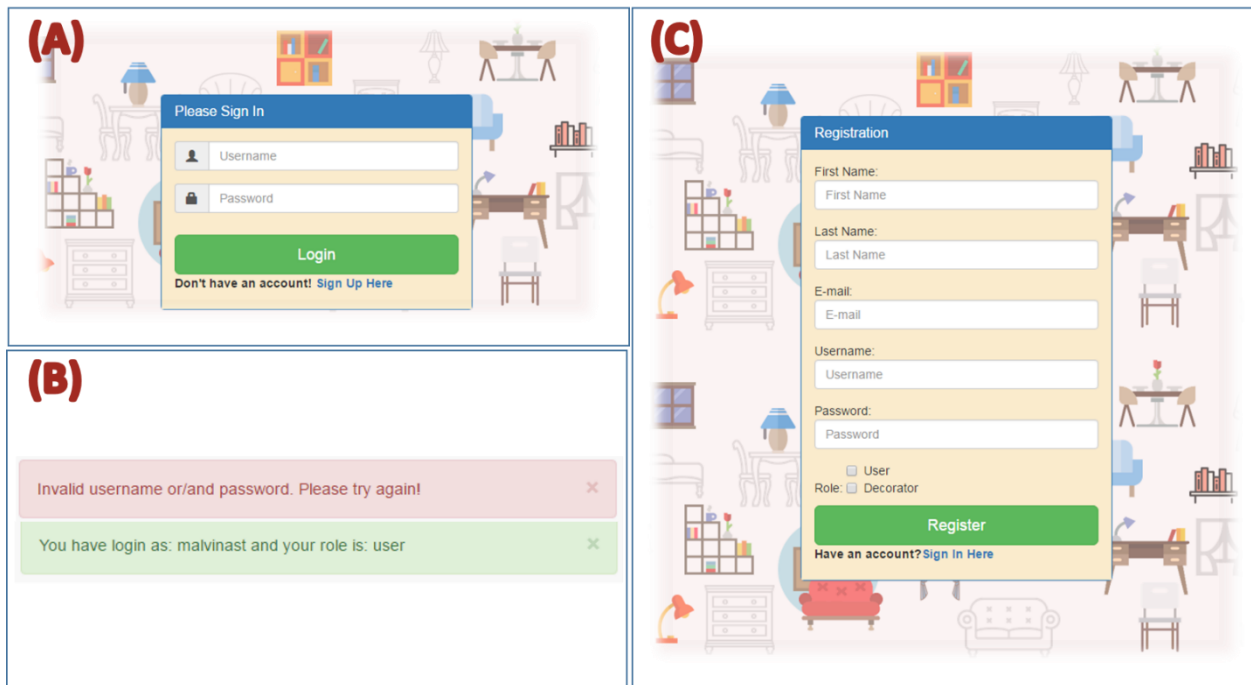


Figure 6-1. Login and Registration layout

The two roles of our application have the following functionalities:

<i>Functionalities</i>	Decorator	User (Customer)
<i>Start a session</i>	✓	✗
<i>Join a session</i>	✗	✓
<i>Show video from all connected users</i>	✓	✓
<i>Chat with all connected users</i>	✓	✓
<i>Complete a questionnaire</i>	✓	✗





















<i>Preview decoration choices</i>		
<i>Insert the selected choice in 3D representation</i>		
<i>Adding recommended items from a list</i>		
<i>Upload an x3d file inside the 3D scene</i>		
<i>Move items around the 3D scene</i>		
<i>Change 3D items' color/texture</i>		
<i>Scale/Rotate 3D items</i>		
<i>Save the 3D scene in x3d format</i>		
<i>Take screenshot from 3D scene</i>		
<i>Close a session</i>		

Table 6-1. Fuctionalities of DECO application

6.1.2 “Session” Manager

After the successful login to our system the users have the ability to join a “session” which is available at the current time. “Session” is synonym with “Group” and becomes available when a decorator selects an unoccupied “session” from a list (see Figure 6-2). On the other hand, if the “session” is occupied, it is appeared in users’ dashboard. Whenever a user or a decorator clicks on “join” button, he is registered to the “session” and has the ability for multi-user conferencing with video and audio call. Moreover, both of them can modify a three-dimensional decoration environment according to Table 6-1. Everyone has the ability to check how many users are in each “session” and who the decorator of this “session” is. Furthermore, our system informs the other decorators if a “session” is already occupied.

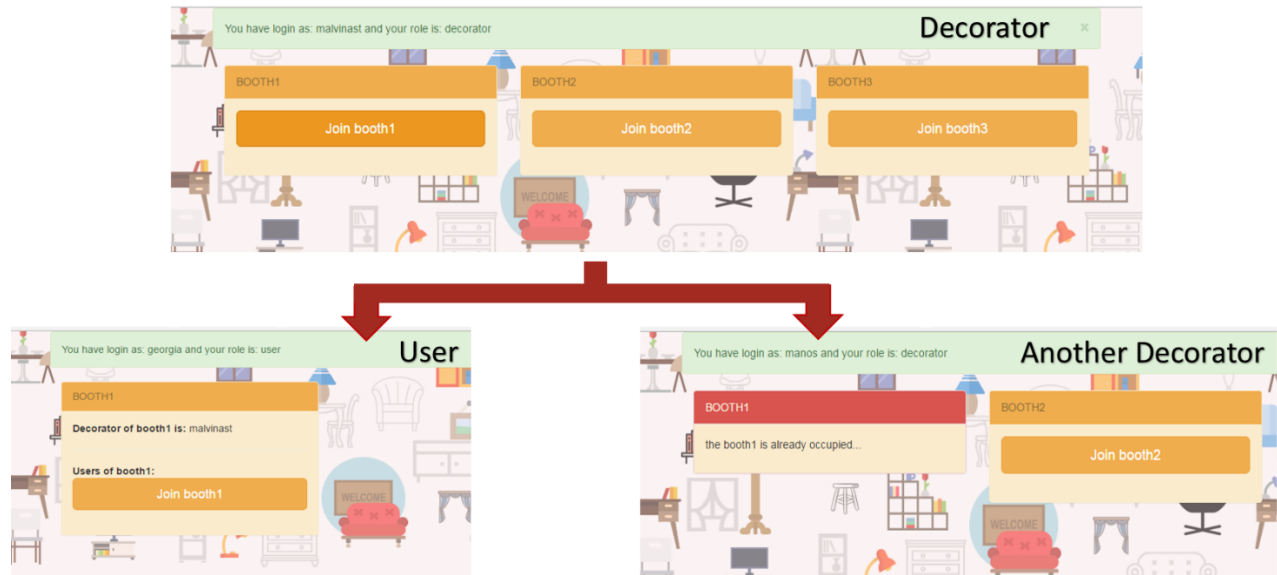


Figure 6-2. Session process

Figure 6-3 shows the "Join" button to each available session, along with a notification message that informs the user for his previously joined sessions. Thanks to this retainment mechanism, our application is able to keep a history of the successfully connected users. So, if for any reason someone logs out by mistake or due to a system fault, he is still able to join the same session again (Figure 6-3C). However, if a decorator logs out, then the session is automatically terminated (Figure 6-3E).

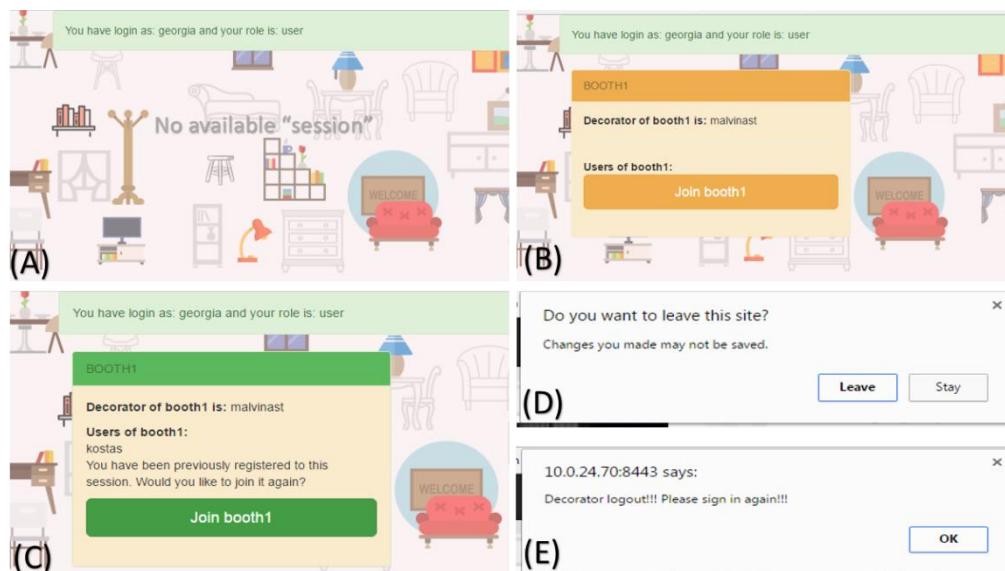


Figure 6-3. Notification messages for user

6.1.3 Content Manager

The first module of the system involves the population of the knowledge base with room spaces. Content providers, may use the web-based interface to assign individuals to classes and define their properties. The object's characteristics such as its material, color and design style can be selected from the ontology. This mechanism provides the ability of storing X3D models for each object in order to build virtual worlds in the next steps. The front-end of the application was implemented in HTML, where the users are navigated through a series of forms, in order to efficiently annotate a room space. The concepts that were used for the room space annotation and decoration are based on the DECO ontology presented in Chapter 3, which has been totally fused into the application.

The second module of the system takes the form of a questionnaire which is based on the stored ontology, using Jena and SDB. Users are asked by the decorator, a number of questions regarding various parameters of the room. The query results are sent back to the decorator as potential decoration solutions matching the needs of the customers. After this, the decorator and the user have the ability for 3D visualization of the selected from the decorator, room space. Our aim was the extension of the online interior design with the choices that we can draw from the fusion of semantic ontologies with Web3D graphics.

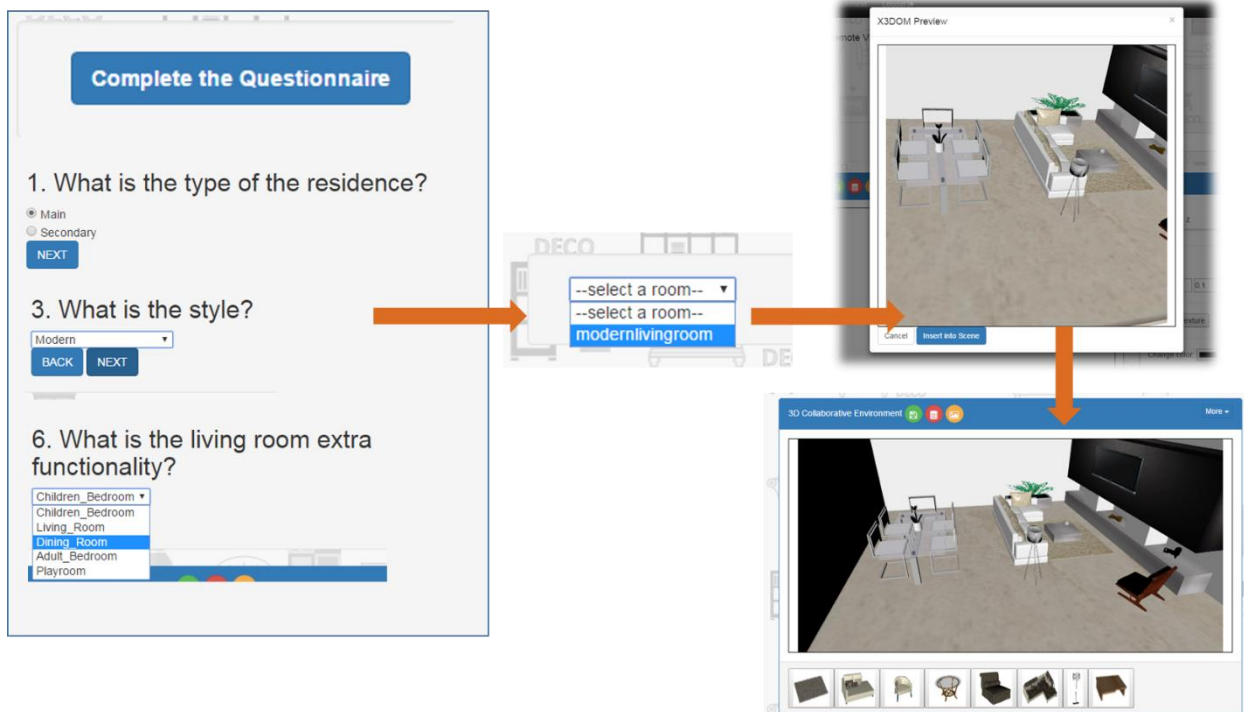


Figure 6-4. Questionnaire and pop-up windows for 3D representation

6.1.4 Collaborative Design Manager

In order to establish our collaborative design manager, we have developed a 3D modeling environment with web-based communication capabilities. Users of the same “session” are working together on their corresponding web pages, sharing the same 3D scene and having the ability to add, remove, change position, color and choose material of the 3D models. In this system, any changes that occur in the scene of a single user are directly transmitted to other users, making feasible the observation of the overall progress in real-time. Our system can be proved a quite useful tool for non-professional customers and simple users.

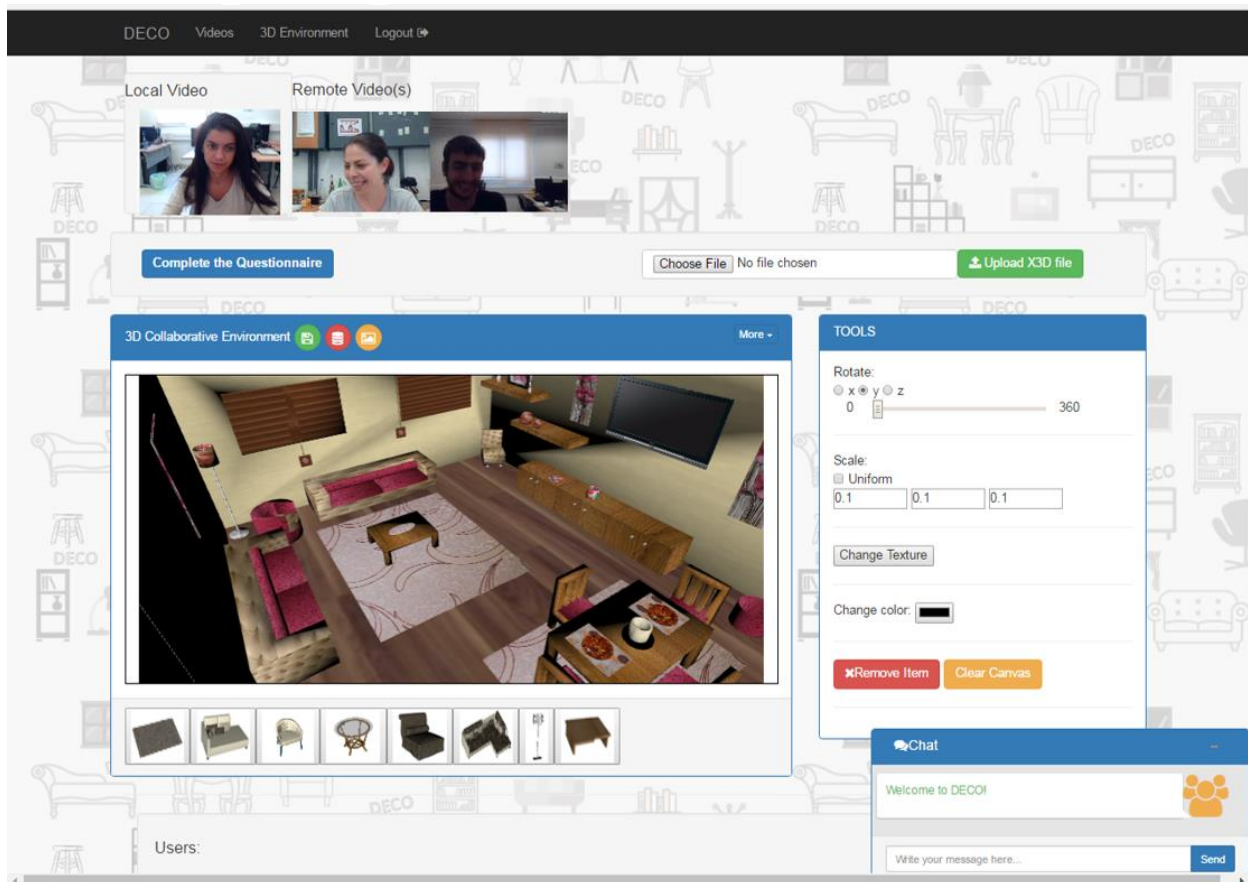


Figure 6-5. DECO application layout

We have mentioned before that our collaborative application has two distinct roles (see Subchapter 6.1.1) known as the “decorator” and the “user”. The decorator of the “session” makes use of the semantic web questionnaire to ask the user a set of predefined questions with video call or chat messages.

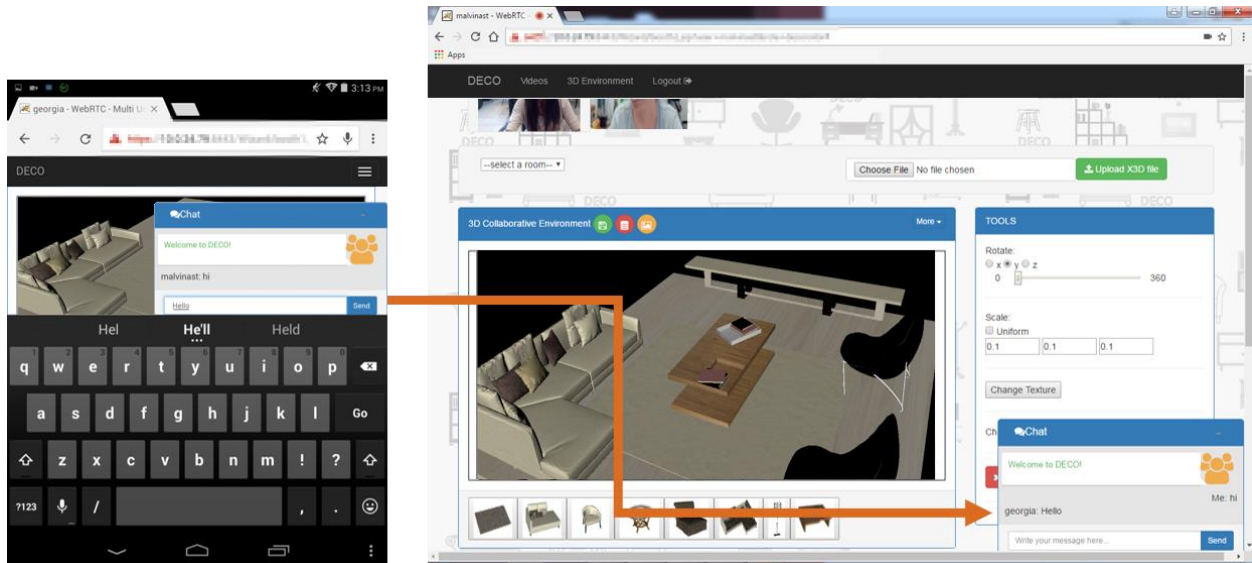


Figure 6-6. DECO Multi-user chat module. Left: tablet view, Right: desktop view

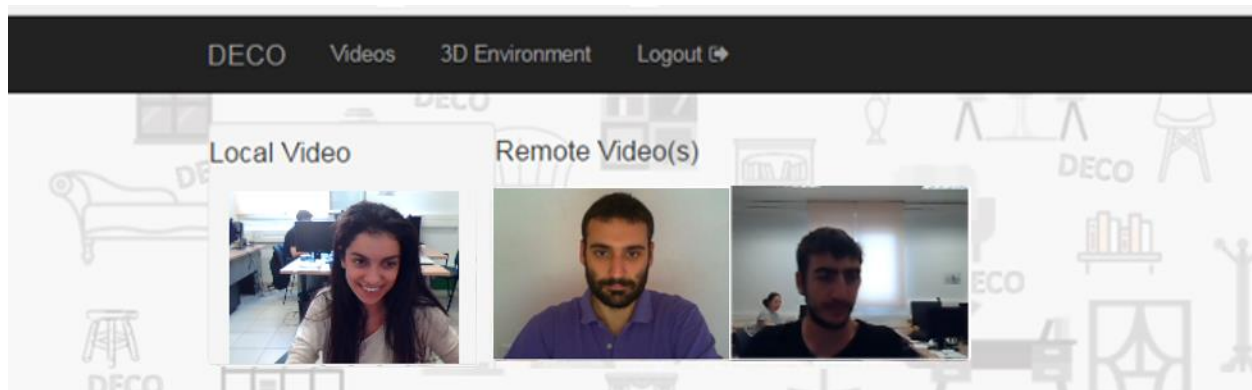


Figure 6-7. DECO Multi-user video call

User's answers are mapped to the room-spaces contained into our knowledge base in order to retrieve and recommend only the most suitable. The chosen ones are displayed to decorator's dashboard in order to choose the room space that is closest to the user's needs. Afterwards, a 3D representation of this room is shown in the webpage, without the use of any plugin or software. At this point, decorator and users of the same "session" share the same 3D scene, allowing all connected users to see each other's actions in real-time. Moreover, the video call feature provides a sufficient conversation mechanism between participants and allows the thorough explanation about what exactly the user wants for his interior space.

Below, we describe how a user may change the color of a specific object and how the rest of connected users are notified about these changes. Generally speaking, the user selects a 3D item from the X3D scene, afterwards he clicks on the color button from the relative menu, and finally he chooses the desired color from the popped up color picker. This selection automatically notifies all other connected users of the same “session” about these changes in their X3D scene.

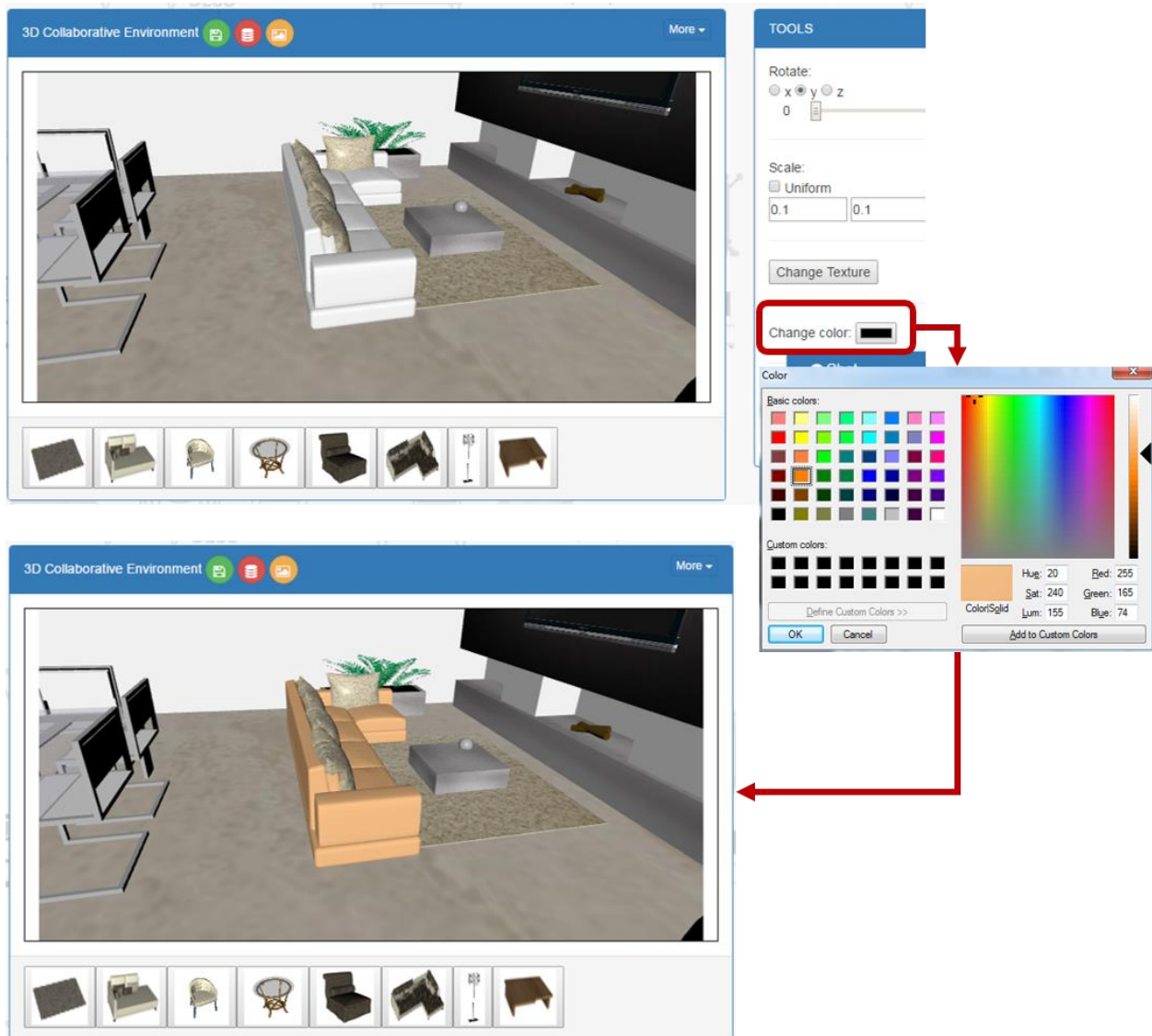


Figure 6-8. Changing the color of an item in X3DOM scene

In the same way, we are able to change the texture, size, position and viewpoints of the corresponding X3D Scene.

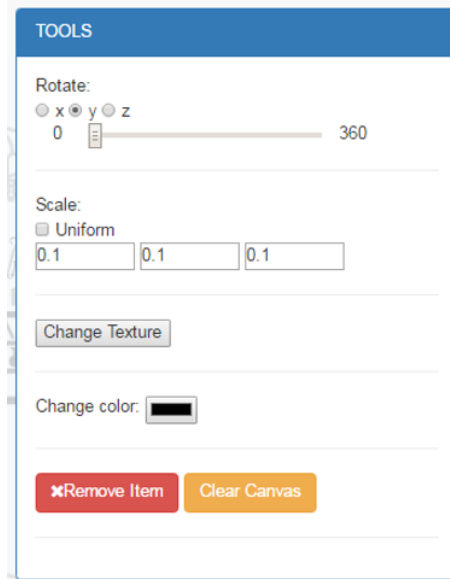


Figure 6-9. DECO tools

Additionally, our application provides extra functionalities to its users, allowing them to insert other 3D items -which are carefully proposed from an ontology- or to delete those items from the scene. The decorator has also the possibility to upload an x3d model.

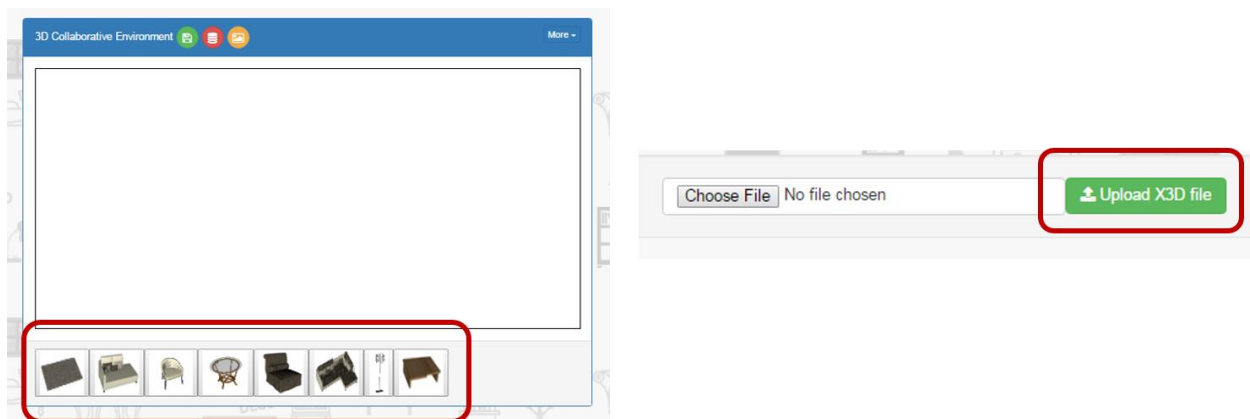


Figure 6-10. Inserting and uploading 3D items

Users may also save screenshots or .x3d files locally to their computers, along with a typical 3D representation of the designed interior space. The decorator can save the 3d representation in the database

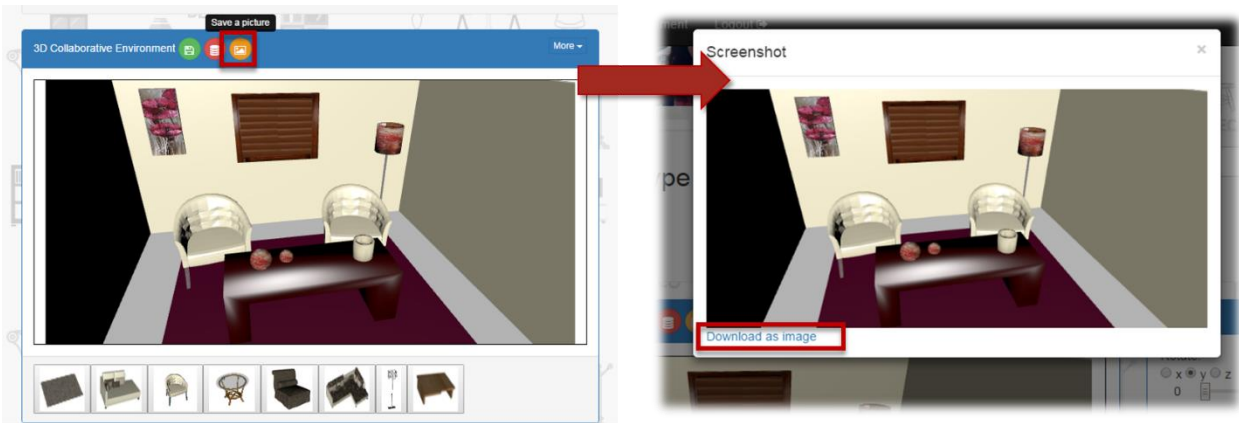


Figure 6-11. Saving a picture in DECO framework

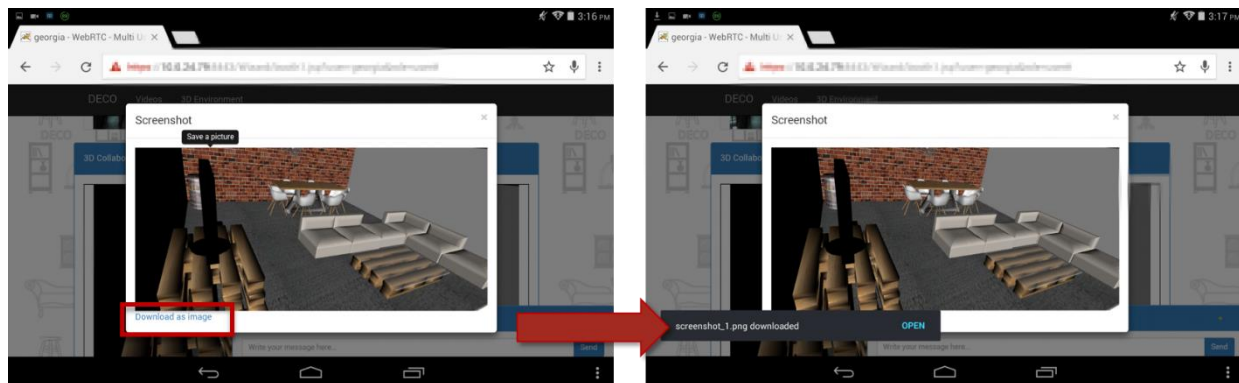


Figure 6-12. Saving a picture in a tablet device

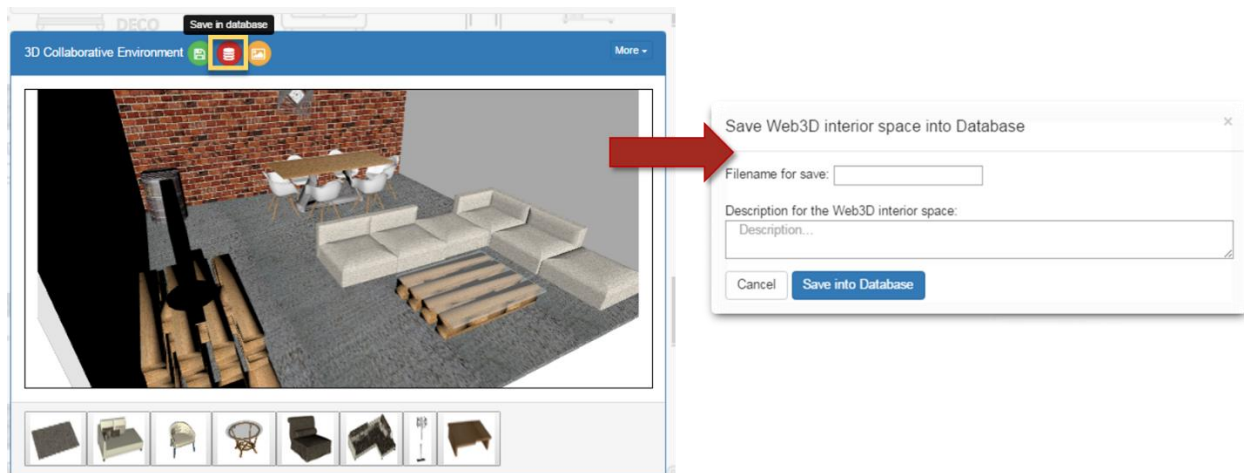


Figure 6-13. Saving the X3D code in a database with DECO

6.2 Technical Design

6.2.1 Login and Registration Manager

For the user management service, we make use of a MySQL database which cooperate with Apache Tomcat and Node.js servers. Whenever user attempts to login or register to Deco system, the corresponding servlets are called and run within Apache Tomcat server. The whole process of registration and authentication is based on servlet methods, which are written in Java language and communicate with our database. On the other hand, the logout feature of the “session” takes place with MySQL module from the Node.js server. This is the best better technique solution for notifying application’s users that the decorator of the specific “session” has left and this “session” is about to close.

```
CREATE TABLE `users` (  
  `id` int(11) NOT NULL AUTO_INCREMENT,  
  `fisrtname` varchar(45) NOT NULL,  
  `lastname` varchar(45) NOT NULL,  
  `email` varchar(45) NOT NULL,  
  `username` varchar(45) NOT NULL,  
  `password` varchar(45) NOT NULL,  
  `role` varchar(45) NOT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Table 6-2. Table users in MySQL

Table 6-2 below represents the structure of users’ table. As we can see, this table consists of seven columns. The first column is the “*id*” of the user and automatically increases its value by one for each new record added to the database. The other columns hold personal information like first name, last name, email, username, password and role of the current user. The column named “*role*” is particularly useful for the system, since its value allows Deco to provide the required functionalities to users.

Figure 6-14 demonstrates the diagram with login and registration steps required for Deco system. The application reads the values from login and registration forms with JavaScript functions and calls the related servlets with the assistance of an AJAX engine. In this way, our system updates successfully parts of the web page without reloading whole page.

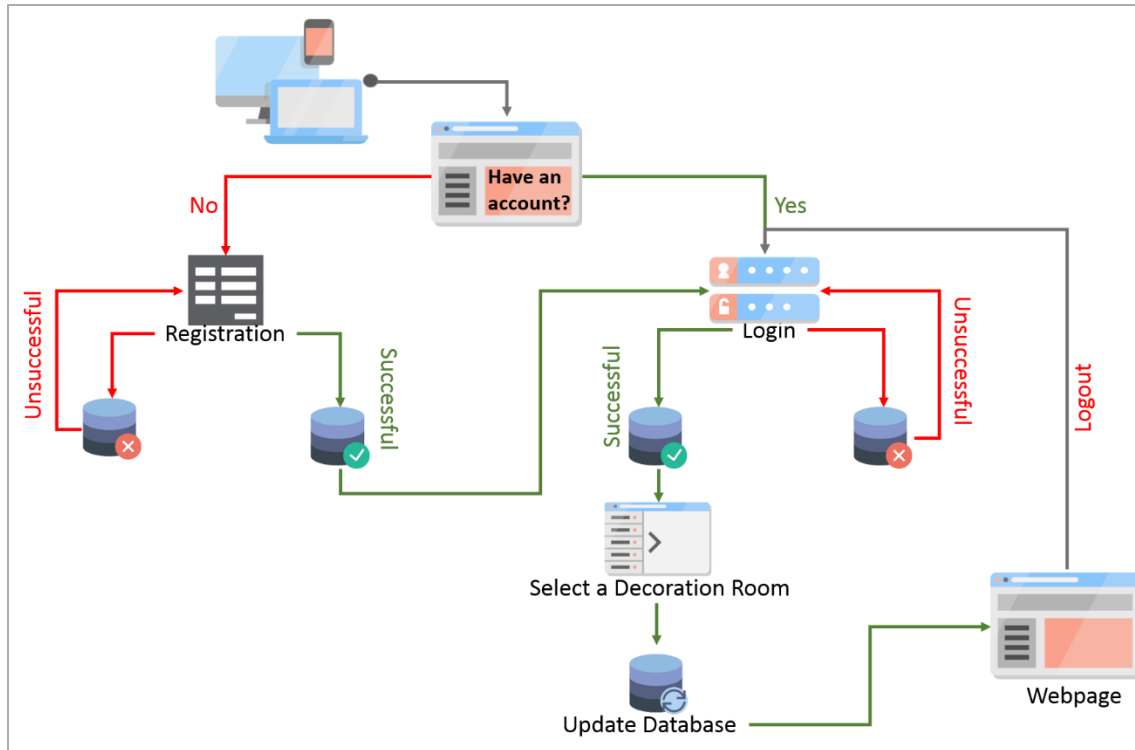


Figure 6-14. Login and Registration System diagram

Login process sequentially executes the following steps:

- At first, the implemented system awaits for user's credentials to be input into the appropriate form
- Afterwards, AJAX POST is used to send the data to server side and more specifically to Login servlet
- At this point, server checks for the authentication of the user by comparing the received data with the stored credentials in MySQL database table and sends the response in JSON format
- Finally, AJAX receives the response from the server and updates the web page without reloading the page

Registration process sequentially executes the following steps:

- At first, the implemented system awaits for user's information to be input into the appropriate form

- Afterwards, AJAX POST is used to send the data to server side and more specifically to Registration servlet
- At this point, server inserts a new record in the “Users” MySQL database table and sends the response in JSON format with successful or unsuccessful registration.
- Finally, AJAX receives the response from the server and updates the web page without reloading the page

6.2.2 Session Manager

The session manager module is responsible to update the table “booths” of our database with the names of the decorator and the users which are login to the specific “session”. The MySQL database is updated with Java servlets after the successful client’s request with AJAX. Also the session manager searches the table “booths” to inform the new users how many users are in this “session” and who the decorator of this “session” is.

Table 6-3 shows the structure of the database

```
CREATE TABLE `booths` (
  `id` int(10) NOT NULL AUTO_INCREMENT,
  `boothname` varchar(45) NOT NULL,
  `url` varchar(150) NOT NULL,
  `user` varchar(150) NOT NULL,
  `decorator` varchar(45) NOT NULL,
  `occupied` tinyint(1) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Table 6-3. Table ‘booths’ in MySQL

6.2.3 Content Manager

Web Ontology Editor Module

The application was developed with Jena framework and takes advantage of its various integrated APIs for the manipulation of DECO ontology. The whole process of room annotation is exclusively done from Jena framework, while the presentation and user interaction is assisted by various technologies like Java Servlets, JSP, JavaScript and CSS. The viewer is capable to explore any ontology on the fly. See chapter 3.6 for more details.

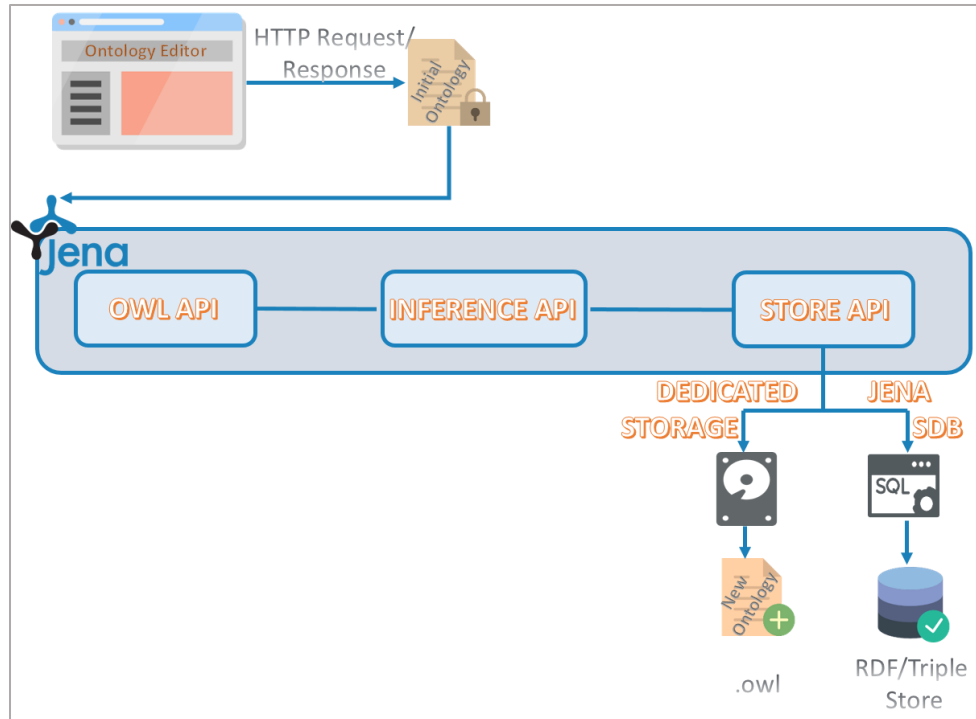


Figure 6-15. DECO architecture for semantic storing

Semantics

We have implemented a mechanism for the retrieval of the room spaces which meet some predefined range of criteria, displayed in the form of a simple HTML questionnaire. The available options to the fields of the questionnaire derive from the "master ontology", which is sourced from the MySQL database and afterwards is transformed into the corresponding RDF graph. The user is granted with the ability to fill out this questionnaire according to his desires and submits it to the system for further processing. The choices made by the user are treated as a single SPARQL query which is processed by the SPARQL API of Jena Model Interface. After that, a connection to MySQL database is established where the SPARQL query is translated to the corresponding SQL query in order to be leveraged from the SQL write operations. Finally, the query is executed against all the existing room spaces in MySQL database, which in turn, returns an HTML form containing these room spaces that fulfill the requirements met by the user.

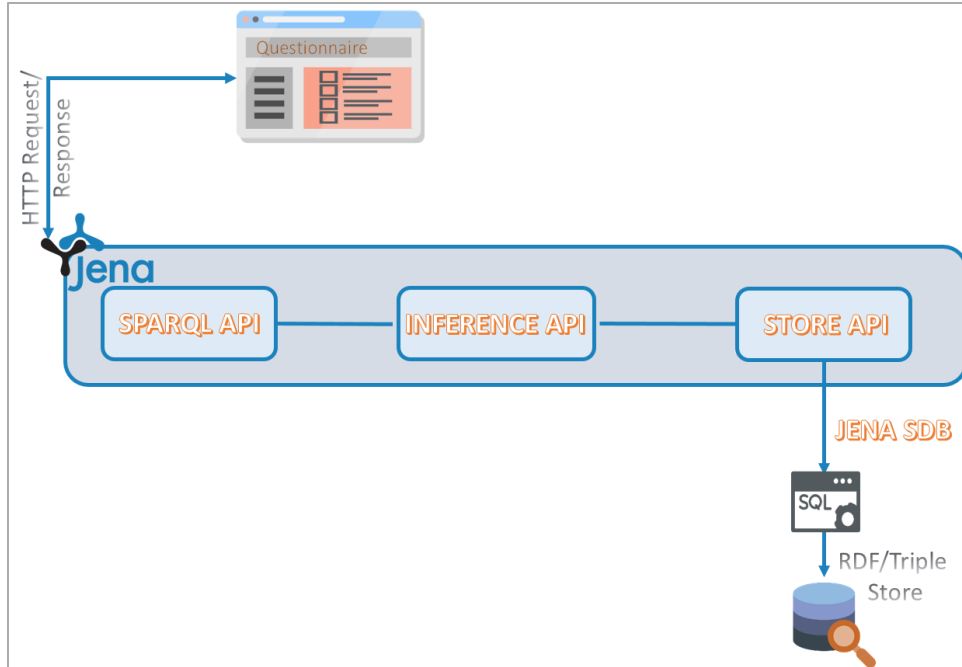


Figure 6-16. DECO architecture for SPARQL engine

6.2.4 Collaborative Design Manager

MCU - Real Time Communication

The MCU server is written in JavaScript language, allowing a wide range of devices (laptop with any operating system, smartphones, tablets with internet connection, etc.) to take advantage of our web application. The application ensures data integrity through a secure channel of communication via HTTPS protocol and guarantees the connectivity under different networks with the collaboration of a STUN server. The whole application is running under Apache Tomcat Server in cooperation with node.js server and socket.io module for the real-time communication part of our application. For this thesis, we merge Apache Tomcat that runs Java based applications with node.js server that runs JavaScript based applications. The socket.io module is used for real time communication. See Chapter 5.4 for more details.

Collaborative Web3D

Our application extends the real-time communication, with the addition of the 3D representation and the extra functionalities that X3DOM framework provides to our users for real

time modification of a room space. The 3D representation is produced after semantic web queries to stored ontologies.

The real-time communication is enabled with socket.io events. Below, how the server manages the messages from this user and how notifies the other users. For the color picker example in Subchapter 6.1.4 section a message is emitted from this user to the node.js server and socket.io event. The server after this broadcasts the received data to the other users. For the real-time communication, it is used the "emit" function of the socket.io as we can see in the last line of code. This line notifies the server that one of the users has change the item with the specific id and color. The data is encoded with JSON format (see Table 4-4).

```
socket.emit('changeColor', {'type': 'changeColor', 'object':  
  lastSelectedObject.getAttribute("id"), 'color': newCol});
```

Table 6-4 presents the code for the server-side. The server receives the “changeColor” event with the JSON data and with the use of the broadcast flag, emits this event with its data to all connected users.

```
socket.on('changeColor', function (data) {  
  socket.broadcast.emit('changeColor',{type:data.type,object:data.object,color  
  :data.color});  
});
```

Table 6-4. Event for changing color from MCU to connected clients

Finally, Table 6-5 presents the code for the other users of the same group. All the users receive the JSON data and update their 3D scene and specifically the 3D item with the new color.

```
socket.on('changeColor', function (data) {  
  var selObj = document.getElementById(data.object);  
  for (child in selObj.childNodes) {  
    if (x3dom.isa(selObj.childNodes[child]._x3domNode,  
    x3dom.nodeTypes.Appearance)) {  
      selObj.childNodes[child]._x3domNode._cf.material.node  
      ._xmlNode.setAttribute('diffuseColor', data.color);  
      x3dom.reload();  
    }  
  }  
});
```

Table 6-5. Listening of ‘changeColor’ event from clients

The previous is an example for the color change in the X3DOM framework with the use of the socket.io events for the real-time communication. In Figure 6-17 is displayed the flowchart of the communication between each user and the server for the previous example. As soon as he chooses the color he likes, a message is emitted from this user to node.js server and a socket.io event is generated. Finally, the server broadcasts the initially received data to the rest of users encoded with JSON format.

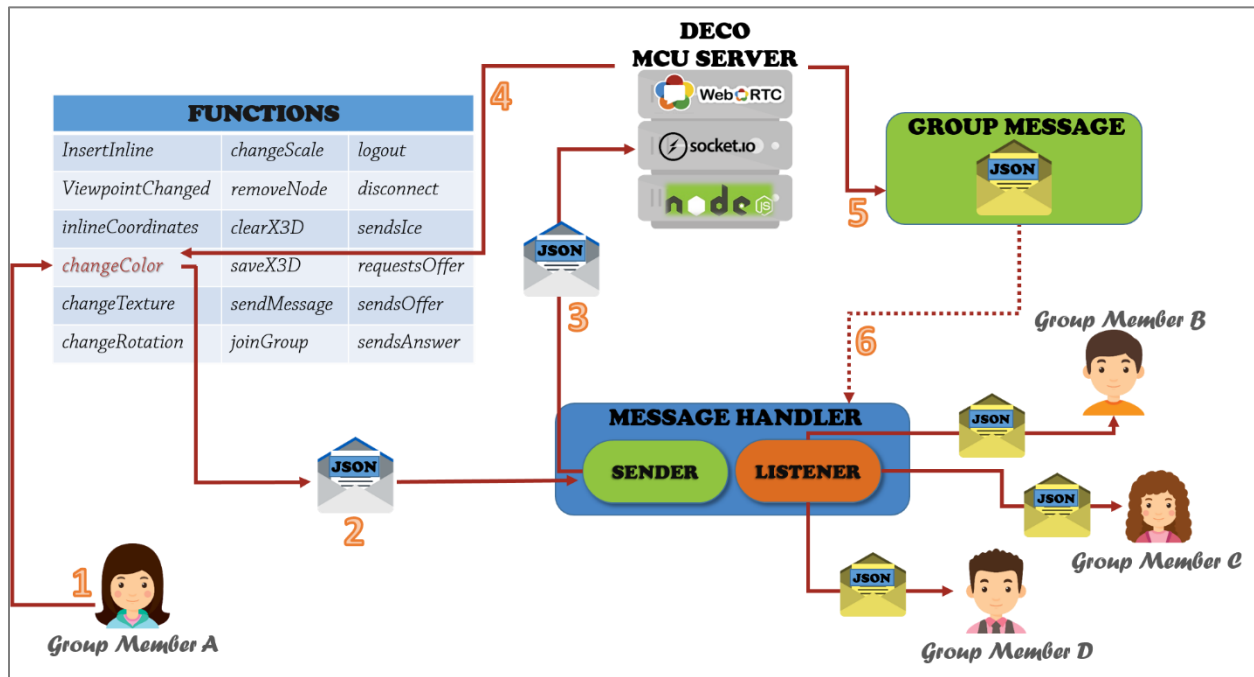


Figure 6-17. Process for real-time communication

Multi- Conferencing Module

We create functions to initiate the signaling process with the creation of a new RTCPeerConnection object for each user who sign in. We keep a list for all connected peers and we manage the action that the server must be done according to the method that every connected client call. Our MCU server detects and manages the events for offer, answer and ice candidate, as we show in the Table 6-6

```

var createUser = function(socket) {
var objOfUser = {'socket': socket};

```



```

//client sending offer
objOfUser.socket.on('userOffer', receivedOffer.bind(objOfUser));
//receiving an answer
objOfUser.socket.on('userAnswer', receivedAnswer.bind(objOfUser));
// listen for ice candidates
objOfUser.socket.on('userIce', receivedIce.bind(objOfUser));

function receivedOffer(data){...}

function receivedAnswer(data){
    var userId = data.userId;
    data.userId = this.getUserId();
    if(connectingUsers[userId]){
        connectingUsers[userId].socket.emit('sendsAnswer', data);
    }
}

function receivedIce(data){...}
...
// Return an instance of the createUser
return objOfUser;
};

```

Table 6-6. MCU code for managing events for multi-user conferencing

On the other side, clients have the following code for the managing of the events

```

var objOfUser = {socket: socket};
// listen for server requesting offer
objOfUser.socket.on('requestsOffer', returnOffer);
// detect receiving an offer from server
objOfUser.socket.on('sendsOffer', returnAnswer);
// receive answer for server
objOfUser.socket.on('sendsAnswer', receiveAnswer);
// listen for ice candidates
objOfUser.socket.on('sendsIce', receiveIce);

// Receive offer request and return an offer
function returnOffer(data){
    // create RTCPeerConnection
    createRTCPeer(function(rtcPeerConnection){
        initializeIce(rtcPeerConnection, data);
        connectingUsers[data.userId] = rtcPeerConnection;
        // create offer and send to server
        rtcPeerConnection.createOffer(
            function(offerResponse){
                data.offer = offerResponse;
                rtcPeerConnection.setLocalDescription(offerResponse,
                    function(){
                        socket.emit('userOffer', data);
                    });
            });
};

```

```

    },
    function() {
        console.log("failure");
    },
    {
        optional: [],
        mandatory: {
            OfferToReceiveAudio: true,
            OfferToReceiveVideo: true
        }
    }
    });
}, false);
function returnAnswer(data) {...}

function receiveAnswer(data) {...}

function receiveIce(data) {...}

...

```

Table 6-7. Clients code for managing the events for multi-user conferencing

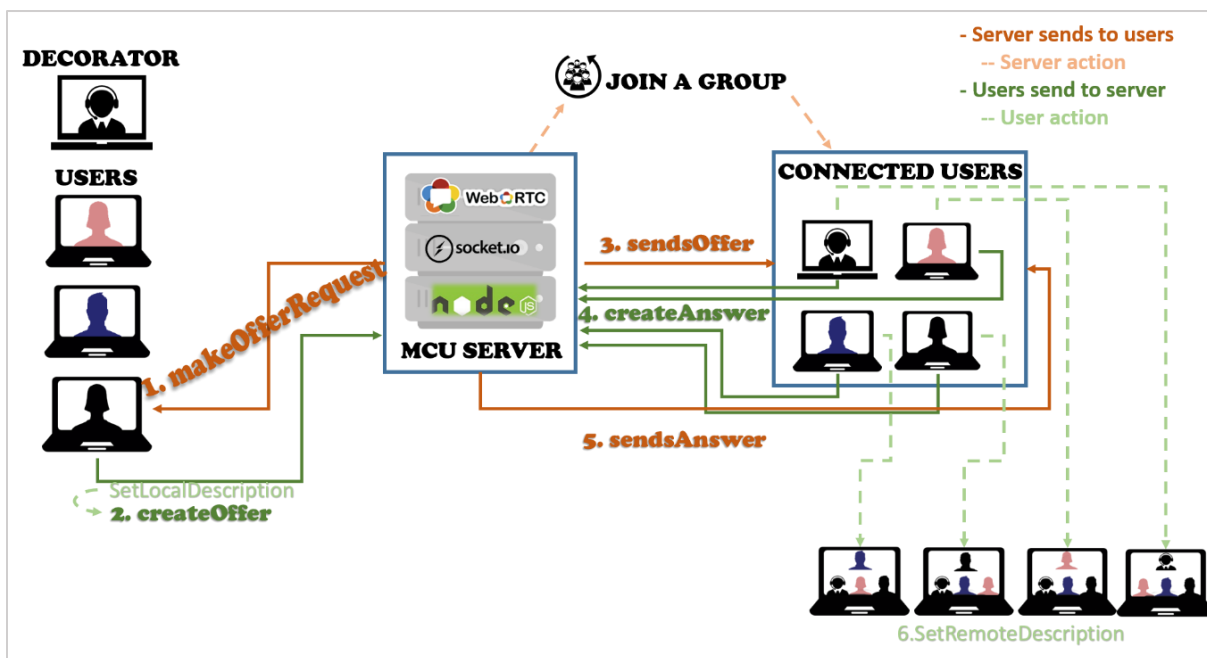


Figure 6-18. Process for multi-user conferencing using our MCU

The whole process is consisting of the steps that are implemented in Figure 6-18. In more detail when a user accesses the webpage the CreateUser function is called and group list with connected users are generated with an RTCPeerConnection for this user, a message from a user is send it to the MCU server with the IceCandidates. A server then, requests an offer for the

specified user and the user responds with the createOffer function and his LocalDescription. When a server receives the offer, emits the data to the connected users of this group. After this, the users response with the createAnswer function of the WebRTC to the MCU server and finally the server emit a messages with all answers to the remote users and the remote users call the SetRemoteDescription function to present the other remote streams to their webpages.

For the connection under different networks, we use a list of free stun servers. We are calling RTCPeerConnection with configuration as parameter. The configuration has the information to access the servers used by ICE which act as STUN servers. The snippet of code in Table 6-8 illustrates the creation of the RTCPeerConnection and the onaddstream function for the remote videos.

```
var remoteDivVideo = document.getElementById('remoteDiv');
function createRTCPeerConnection(setConnection, isAnswer){
    var RTCPeerConnection = window.RTCPeerConnection ||
    window.mozRTCPeerConnection || window.webkitRTCPeerConnection;
    var ICE_config= {
        'iceServers': [
            {url:'stun:stun01.sipphone.com'},
            {url:'stun:stun.ekiga.net'},
            {url:'stun:stun.fwdnet.net'},
            {url:'stun:stun.ideasip.com'},
            {url:'stun:stun.iptel.org'},
            {url:'stun:stun.rixtelecom.se'},
            {url:'stun:stun.schlund.de'},
            {url:'stun:stun.l.google.com:19302'},
            {url:'stun:stun1.l.google.com:19302'},
            {url:'stun:stun2.l.google.com:19302'},
            {url:'stun:stun3.l.google.com:19302'},
            {url:'stun:stun4.l.google.com:19302'},
            {url:'stun:stunserver.org'},
            {url:'stun:stun.softjoys.com'},
            {url:'stun:stun.voiparound.com'},
            {url:'stun:stun.voipbuster.com'},
            {url:'stun:stun.voipstunt.com'},
            {url:'stun:stun.voxgratia.org'},
            {url:'stun:stun.xten.com'},
            {
                url: 'turn:numb.viagenie.ca',
                credential: 'muazkh',
                username: 'webrtc@live.com'
            },
            {
                url: 'turn:192.158.29.39:3478?transport=udp',
                credential: 'JZE0Et2V3Qb0y27GRntt2u2PAYA=',
                username: '28224511:1379330808'
            },
            {
                url: 'turn:192.158.29.39:3478?transport=tcp',
```

```

        credential: 'JZEOEt2V3Qb0y27GRntt2u2PAYA=',
        username: '28224511:1379330808'
    }]);

    var pc = new RTCPeerConnection(ICE_config);

    var remoteStream;
    pc.onaddstream = function(evt){
        var newVideo = document.createElement('video');
        newVideo.className = "video";
        newVideo.width=150;
        newVideo.length=150;
        newVideo.src = URL.createObjectURL(evt.stream);
        remoteDivVideo.appendChild(newVideo);
        newVideo.play();
    };

    pc.onclose = function(){
        if(remoteStream){
            remoteStream.parentNode.removeChild(remoteStream);
        }
    };

    pc.addStream(myStream);
    setConnection(pc);
}

```

Table 6-8. Code for STUN and addstream function

Chapter 7 - Conclusions

Our previous avocation with ontologies and Semantic Web gave birth to an innovative environment that successfully integrated the visual representation and semantic layers with a real-time communication layer between its users. In this way, it was created a standalone product which can be used as the basis for decoration and interior design purposes from decorators and companies. Taking into consideration modern era's capabilities and technologies along with the continuous increment of users navigated to Internet via their mobile devices (e.g. smartphones, tablets, etc.), our work also involved state-of-the-art technologies that are independent of plugins or additional software. Moreover, the Responsive Web Design was applied throughout our application in order to ensure its ease of use and accessibility from any kind of device and platform.

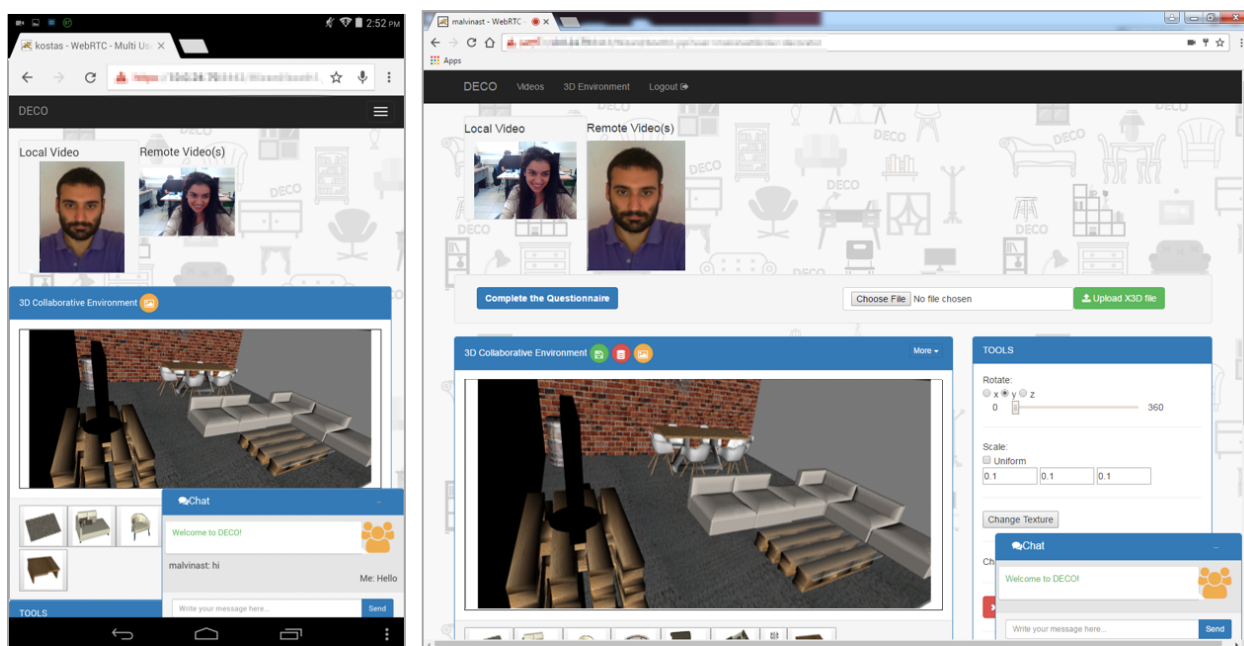


Figure 7-1. Cross-platform and multi-device support

The security of the communication is guaranteed via the HTTPS protocol, while its integrity and stability are backed by a carefully designed Multi-Point Control Unit. Video and audio quality are heavily depending on each user's bandwidth, meaning that user experience is favored by high bandwidth connections. Another important factor for the performance of our

system is the device being used by the user. Devices with advanced graphic cards can load changes in the 3D collaborative part faster than obsolete or older generation graphic cards.

Finally, the target groups of the implemented application do not limit to decorators and interior design companies, but simple users are able to create their own room space according to their likes and interact with a manufacturer or a domain expert to finalize their initial architectural pattern. An X3D scene translated to a real-world room based on an interior design sketch

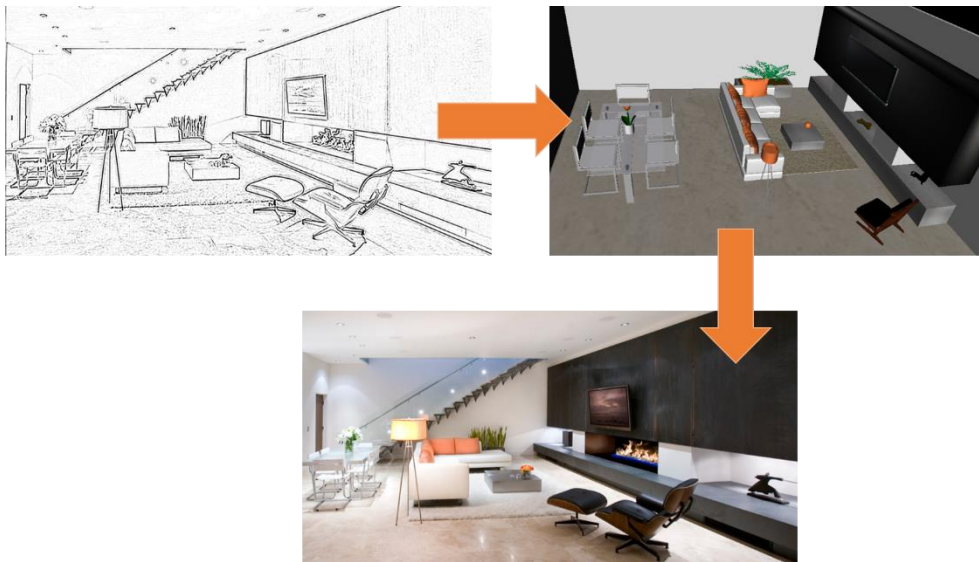


Figure 7-2. Mapping an X3D scene to a real-world room space

References

- [1] "Autodesk HomeStyles," [Online]. Available: <http://www.homestyler.com/home>. [Accessed 10 09 2016].
- [2] "Roomsketcher," [Online]. Available: <http://www.roomsketcher.com/homedesigner/>. [Accessed 2016 09 10].
- [3] "planner 5D," [Online]. Available: <https://planner5d.com/>. [Accessed 08 09 2016].
- [4] "Roomstyler," [Online]. Available: <https://roomstyler.com/3dplanner>. [Accessed 08 09 2016].
- [5] "IKEA home planner," [Online]. Available: http://www.ikea.com/ms/en_US/rooms_ideas/splashplanners_new.html. [Accessed 10 09 2016].
- [6] F. Harmelen and D. McGuinness, "OWL web ontology language overview.," 2004.
- [7] T. Berners-Lee, J. Handler and O. Lassila, The Semantic Web. A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities., vol. 284, Scientific American, 2001.
- [8] G. Stamou and S. Kollias, Multimedia content and the semantic Web., John Wiley & Sons, 2005.
- [9] L. Makris, N. Karatzoulis and D. Tzovaras, "A DIYD (Do It Yourself Design) e-commerce system for vehicle design based on ontologies and 3D visualization.," in *International Conference on Universal Access in Human-Computer Interaction*, 2007.
- [10] N. Karatzoulis, I. Tsampoulatidis, I. Maglogiannis, I. Zormpas, D. Tzovaras and M. Strintzis, "Intelligent Configurable Electronic Shop Platform based on Ontologies and 3D Visualization," in *IFIP International Conference on Artificial Intelligence Applications and Innovations*, 2006.
- [11] I. Tsamoulatidis, G. Nikolakis, D. Tzovaras and G. M. Strintzis, "Ontology based interactive graphic environment for product presentation.," in *Computer Graphics International*.
- [12] J. Lee and Y. Jeong, "User-centric knowledge representations based on ontology for AEC design collaboration," in *Computer-Aided Design*, vol. 44, Elsevier, 2012, pp. 735-748.

- [13] K. Kontakis, M. Steiakaki, K. Kapetanakis and A. G. Malamos, "DEC-O: an ontology framework and interactive 3D interface for interior," in *Proceedings of the 19th International ACM Conference on 3D Web Technologies*, Vancouver, British Columbia, Canada, 2014.
- [14] S. Irawati, D. Calderon and H. Ko, "Spatial ontology for semantic integration in 3D multimodal interaction framework.," in *In Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications*, 2006.
- [15] E. M. Latoschik and M. Schilling, "Incorporating VR Databases into AI Knowledge Representations: A Framework for Intelligent Graphics Applications.," in *Computer Graphics and Imaging*, 2003, pp. 79-84.
- [16] K. Kontakis, "Semantic description of scenes in virtual reality environments.," 30 11 2015. [Online]. Available: <https://apothesis.lib.teicrete.gr/handle/11713/3790>.
- [17] M. Martin, J. Unbehauen and S. Auer, "Improving the Performance of Semantic Web Applications with SPARQL Query Caching," in *The Semantic Web: Research and Applications*, vol. 6089, Springer, 2010, pp. 304-318.
- [18] X. Wang, T. Lv, S. Wang and Z. Wang, "An ontology and SWRL based 3D model retrieval system.," in *Asia Information Retrieval Symposium*, 2008.
- [19] "SPARQL Query Language for RDF," 2008.
- [20] P. Smart, A. Russell, D. Braines, Y. Kalfoglou, J. Bao and N. Shadbolt, "A visual approach to semantic query design using a web-based graphical query designer," in *International Conference on Knowledge Engineering and Knowledge Management*, 2008.
- [21] A. De Leon Battista, N. Villanueva-Rosales, M. Palenychka and M. Dumontier, "SMART: A Web-Based, Ontology-Driven, Semantic Web Query Answering Application," in *Proceedings of the 2007 International Conference on Semantic Web Challenge-Volume 295*, 2007.
- [22] K. Kontakis, M. Steiakaki, M. Kalochristinakis, K. Kapetanakis and A. G. Malamos, "Applying Aesthetic Rules in Virtual Environments by means of Semantic Web Technologies," in *Augmented and Virtual Reality*, vol. 9254, Springer International Publishing, 2015, pp. 344-354.
- [23] A. G. Malamos, V. P. Sympa and G. S. Mamakis, "Xml Annotation Of Conceptual Characteristics In Interior Decoration," in *6th International Conference, New Horizons in Industry, Business and Education (NHIBE)*, 2009.

- [24] K. Kontakis, M. Steiakaki, M. Kalochristianakis and A. G. Malamos, "Applying Aesthetic Rules in Virtual Environments by means of Semantic Web Technologies," in *Augmented and Virtual Reality*, vol. 9254, Springer International Publishing, 2015, pp. 344-354.
- [25] D. Brutzman and D. Leonard, X3D: extensible 3D graphics for Web authors, Morgan Kaufmann, 2010.
- [26] J. Behr, P. Eschler, Y. Jung and M. Zollner, "X3DOM – A DOM-based HTML5/ X3D Integration Model," in *Proceedings of the 14th International Conference on 3D Web Technology*, 2009.
- [27] A. Stamoulias, E. Lakka and A. G. Malamos, "“Wrapping” X3DOM around Web Audio API," *International Journal of Interactive Multimedia and Artificial Intelligence*, vol. 3, pp. 36-46, 2015.
- [28] A. Stamoulias, A. G. Malamos, M. Zampoglou and D. Brutzman, "Enhancing X3DOM declarative 3D with rigid body physics support.," in *Proceedings of the 19th International ACM Conference on 3D Web Technologies.*, 2014.
- [29] K. Kapetanakis, S. Panagiotakis, A. G. Malamos and M. Zampoglou, "Adaptive Video Streaming on top of Web3D A bridging technology between X3DOM and MPEG-DASH," in *Telecommunications and Multimedia (TEMU)*, 2014.
- [30] T. N. Eicke , Y. Jung and A. Kuijper, "Stable dynamic webshadows in the X3DOM framework," *Expert Systems with Applications*, vol. 42, pp. 3585-3609, 2015.
- [31] S. Hughes, P. Brusilovsky and M. Lewis, "Adaptive navigation support in 3D e-commerce activities," in *Proc. of Workshop on Recommendation and Personalization in eCommerce at AH*, 2002.
- [32] L. Chittaro and R. Ranon, "Web3D technologies in learning, education and training: motivations, issues, opportunities," *Computers & Education*, vol. 49, pp. 3-18, 2007.
- [33] R. Rolland, E. Yvain, O. Christmann, E. Loup-Escande and S. Richir, "E-commerce and Web 3D for involving the customer in the design process: the case of a gates 3D configurator," in *In Proceedings of the 2012 Virtual Reality International Conference*, 2012.
- [34] K. Kapetanakis, P. Spala , P. Symba, G. Mamakis and A. G. Malamos, "A novel approach in converting SVG architectural data to X3D worlds.," *In Proceedings of the international conference on telecommunications and multimedia, TEMU*, pp. 14-16, July 2010.

- [35] M. Zampoglou, A. G. Malamos, K. Kapetanakis, K. Kontakis, E. Sardis, G. Vafiadis, M. Vrettos and A. Doulamis, "iPromotion: A Cloud-Based Platform," in *Big Data and Internet of Things: A Roadmap for Smart Environments.*, Springer International Publishing, 2014, pp. 447-470.
- [36] S. Panagiotakis, K. Kapetanakis and A. G. Malamos, "Architecture for Real Time Communications over the Web," *International Journal of Web Engineering 2.1*, pp. 1-8, 2013.
- [37] C. Weiner, "Mozilla Firefox," March 2016. [Online]. Available: <https://blog.mozilla.org/futurereleases/2014/10/16/test-the-new-firefox-hello-webrtc-feature-in-firefox-beta/>.
- [38] S. Panagiotakis, I. Vakintis, H. Andrioti, A. Stamoulias, K. Kapetanakis and A. G. Malamos, "Towards Ubiquitous and Adaptive Web-Based Multimedia Communications via the Cloud," in *Resource Management of Mobile Cloud Computing Networks and Environments*, 2015.
- [39] H. Andrioti, A. Stamoulias, K. Kapetanakis, S. Panagiotakis and A. G. Malamos, "Integrating WebRTC and X3DOM: bridging the gap between communications and graphics," in *Proceedings of the 20th International Conference on 3D Web Technology*, Heraklion, Greece, 2015.
- [40] "WEB REAL-TIME COMMUNICATION 2014 CONFERENCE IN MUNICH," [Online]. Available: <http://webrtc-event.com/>. [Accessed March 2016].
- [41] "Intel Collaboration Suite for WebRTC," [Online]. Available: <https://software.intel.com/en-us/webrtc-sdk>. [Accessed August 2016].
- [42] "Kurento," [Online]. Available: <http://www.kurento.org/>. [Accessed August 2016].
- [43] "Licode," [Online]. Available: <http://lynckia.com/licode/>. [Accessed August 2016].
- [44] "Jitsi," [Online]. Available: <https://jitsi.org/>. [Accessed August 2016].
- [45] "NPM ecosystem - Simple, awesome, asynchronous dependency injection and lifecycle control," [Online]. Available: <https://www.npmjs.com/package/ecosystem>. [Accessed May 2016].
- [46] "Shining Light Productions for Win32 OpenSSL," [Online]. Available: <http://slproweb.com/products/Win32OpenSSL.html>. [Accessed May 2016].
- [47] S. Loreto and S. Romano, *Real-Time Communication with WebRTC: Peer-to-Peer in the Browser.*, O'Reilly Media, Inc., 2014.

- [48] S. Tilkov and S. Vinoski, "Node. js: Using JavaScript to build high-performance network programs.," in *IEEE Internet Computing*, IEEE, 2010.
- [49] "Oracle Corporation - A MySQL Strategy Whitepaper.," 2011. [Online]. Available: <http://www.oracle.com/us/products/mysql/mysql-wp-top10-webbased-apps-461054.pdf>.

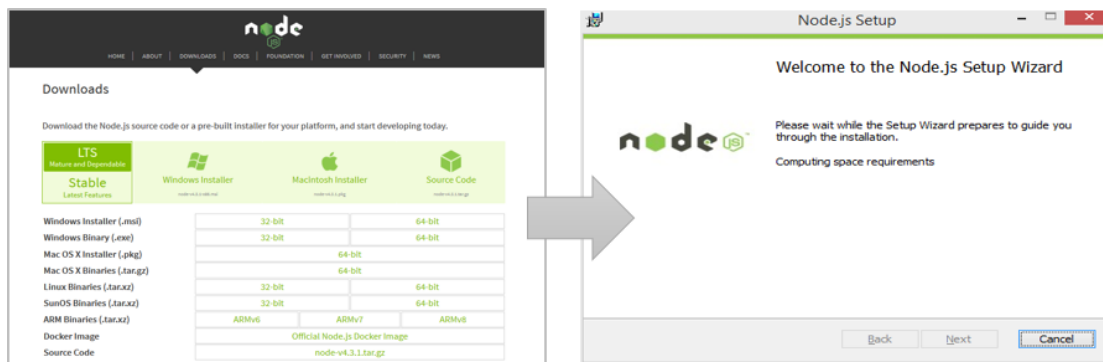
Appendix A - WEBRTC Configuration and programming

This section describes in detail all of the necessary steps for setting up and running a simple chat application using Node.js runtime environment and Socket.io real-time framework. Furthermore, it also includes a brief but concise introduction to generating self-signed SSL certificates in order to satisfy browser security mechanisms, an obstacle met in the “getUserMedia” API of WebRTC protocol.

Configuring Node.js & Socket.io

Node.js is a cross-platform environment especially made for the development of fast, scalable network applications. It started and remained as an open-source initiative for server-side web applications, using a non-blocking I/O model to optimize scalability in real-time applications. Node.js is packed with npm ecosystem, the largest open source library repository that provides advanced asynchronous and lifecycle control capabilities at runtime environment [45]. On the other hand, Socket.io is used for the implementation of real-time applications in Web browsers and mobile devices, enabling synchronized communication mechanisms between multiple users with just a few lines of JavaScript code. The basic steps for configuring these two important modules are briefly described below.

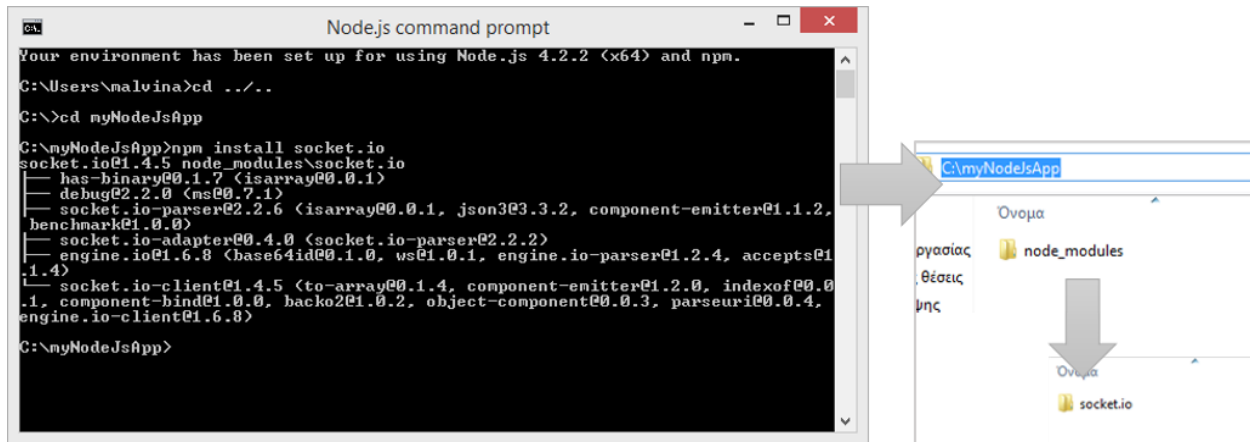
Node.js is publicly available at <https://nodejs.org/en/download/>, where a variety of installers can be chosen according to user’s operating system.



Appendix A - Fig. 1. Node.js installation packages

After the successful installation of Node.js, it has to be determined the folder which is going to host the application. The navigation to this specific folder is feasible through a windows command prompt (e.g. cmd.exe) or Node.js cmd, followed by the execution of the command `npm`

`install socket.io`. Immediately, a `node_modules` folder is created within the selected folder to satisfy application's needs.



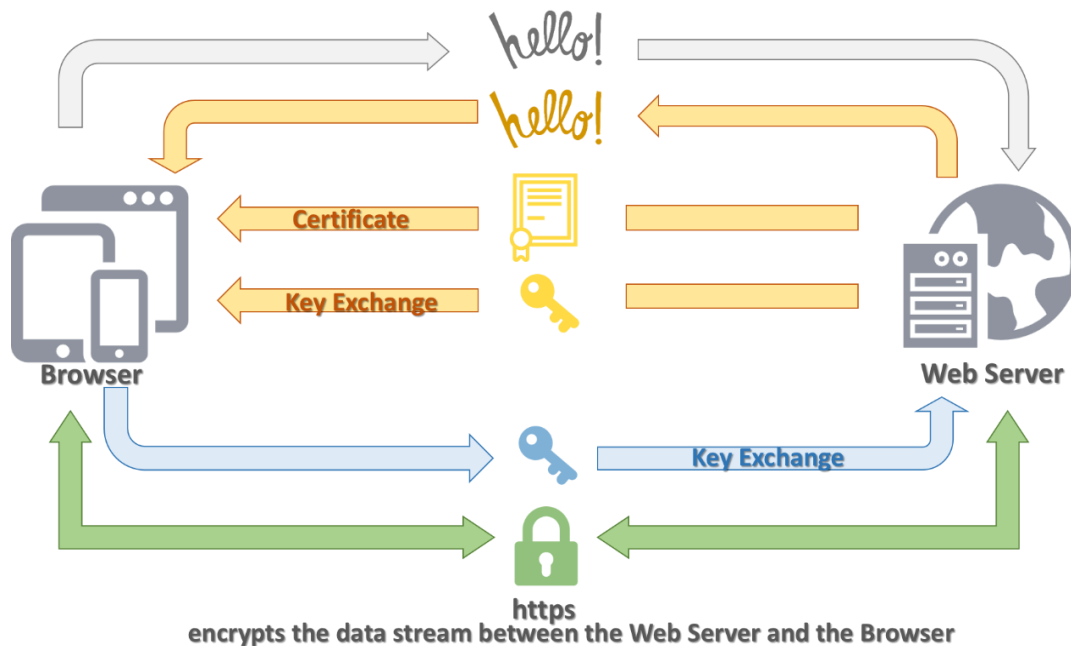
Appendix A - Fig. 2. Installing Socket.io module

Creation of the web server

Till recently, the communication over a computer network was based on an unencrypted protocol. However, on February the 17th 2015, HTTP/2 protocol (e.g. HTTPS) was approved by the IESG as the proposed standard for daily use in web browsing. In other words, this was a major milestone which allowed all servers to make use of HTTP/2. HTTPS is the HTTPs protocol over TLS/SSL which is responsible for exchange of data between clients' browser and a website. The 'S' at the end of HTTPS stands for 'Secure', denoting that within this protocol a connection is encrypted by Transport Layer Security or Secure Sockets Layer. Doing so, the data stream of highly confidential transactions like online shopping and e-banking, are always protected during web server and web client's communication. Appendix A - Fig. 3 displays the handshake sequence that takes place in HTTPS protocol.

At first, the client initiates the communication and server establishes the connection. Afterwards, the client sends a message containing the generated certificate to prove his identity and the symmetric key which is going to be used for encryption and decryption. The server receives both of them and validates these messages' integrity using his own private key. Then, server notifies client that his symmetric key has been successfully accepted for cipher specification transmission and closes the handshake.

The following code enables the HTTPS connection within Node.js server. The first line of code activates the built-in https module, while the second one enables the responsible module for reading -and writing to- files.



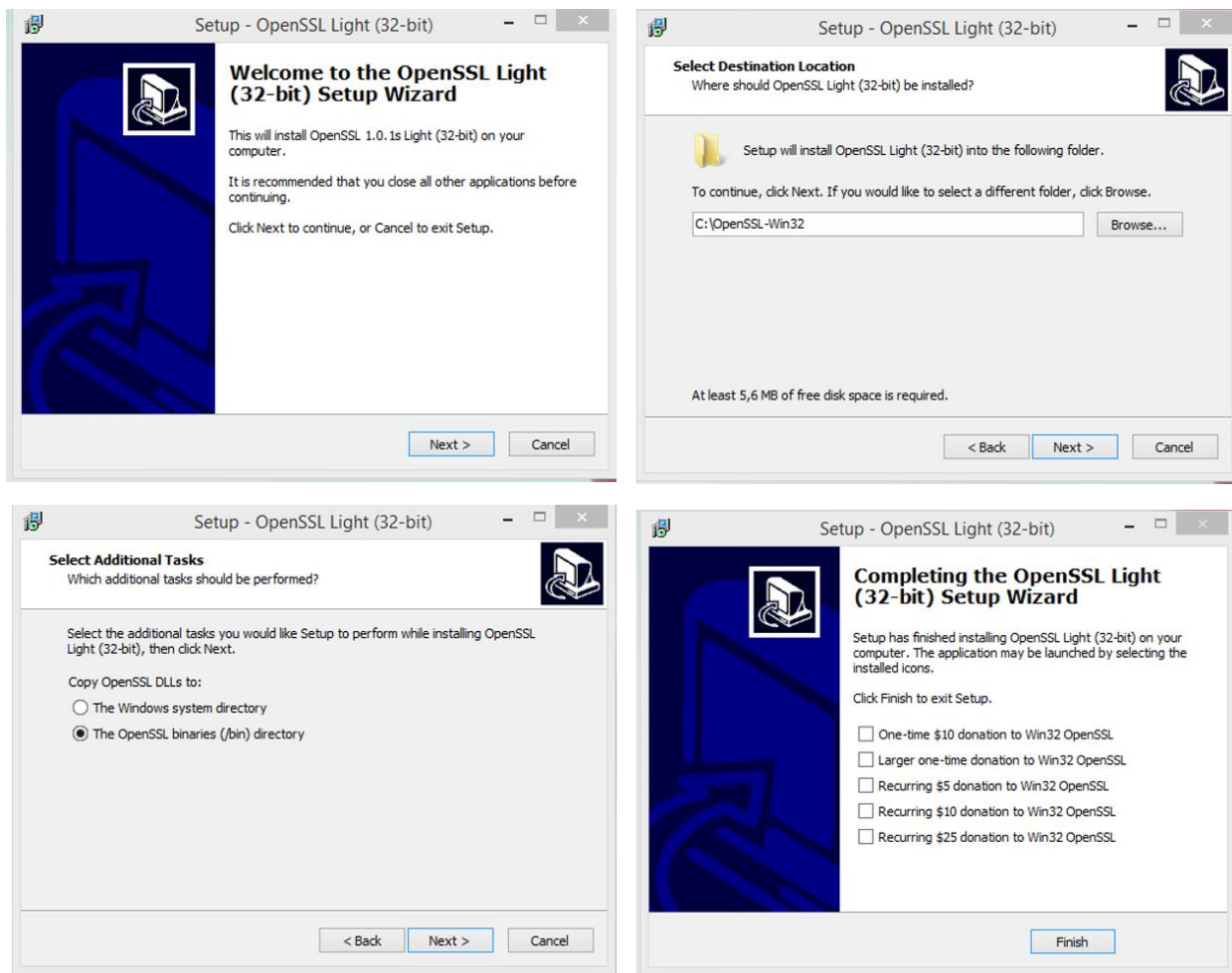
Appendix A - Fig. 4. HTTPS handshake

Generating Certificates with OpenSSL Toolkit

The prerequisites for setting up the HTTPS server met by our web application, additionally require the fulfillment of an SSL certificate and NodeJS built-in https module. Generally speaking, there are two kinds of certificates: those signed by a 'Certificate Authority' known as CA, and those which are 'self-signed certificates'. Concerning the first ones, a Certificate Authority is nothing more than a third-party source offering a trusted SSL certificate that allows users to validate a website's identity. In most cases, such a CA-signed certificate is the perfect candidate for applications deployed on a production environment, since authentication and trust services are of utmost importance for users browsing the Web. However, a self-signed certificate is more than sufficient for development environments and testing purposes.

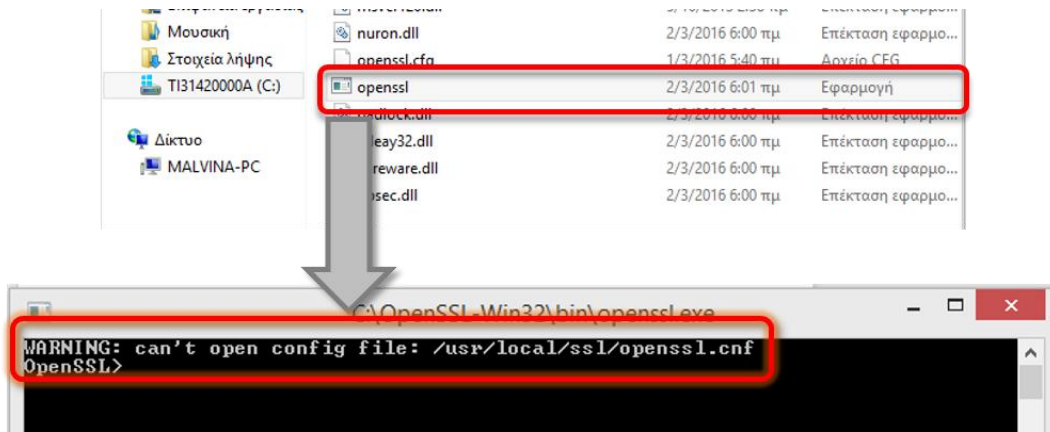
Installing OpenSSL

Even though there are a couple of decent solutions for the creation of self-signed certificates on Linux distributions, Windows platforms are rather poor in manipulating any SSL certificates. For that reason it is suggested the use of an OpenSSL installation package which comes with thoroughly compiling instructions in its corresponding README and INSTALL files. The whole procedure needs Microsoft Platform SDK and Perl under Cygwin or under ActiveState Perl. On the other hand, there are also easier alternative installation methodologies with the latest “Light” version of OpenSSL [46].



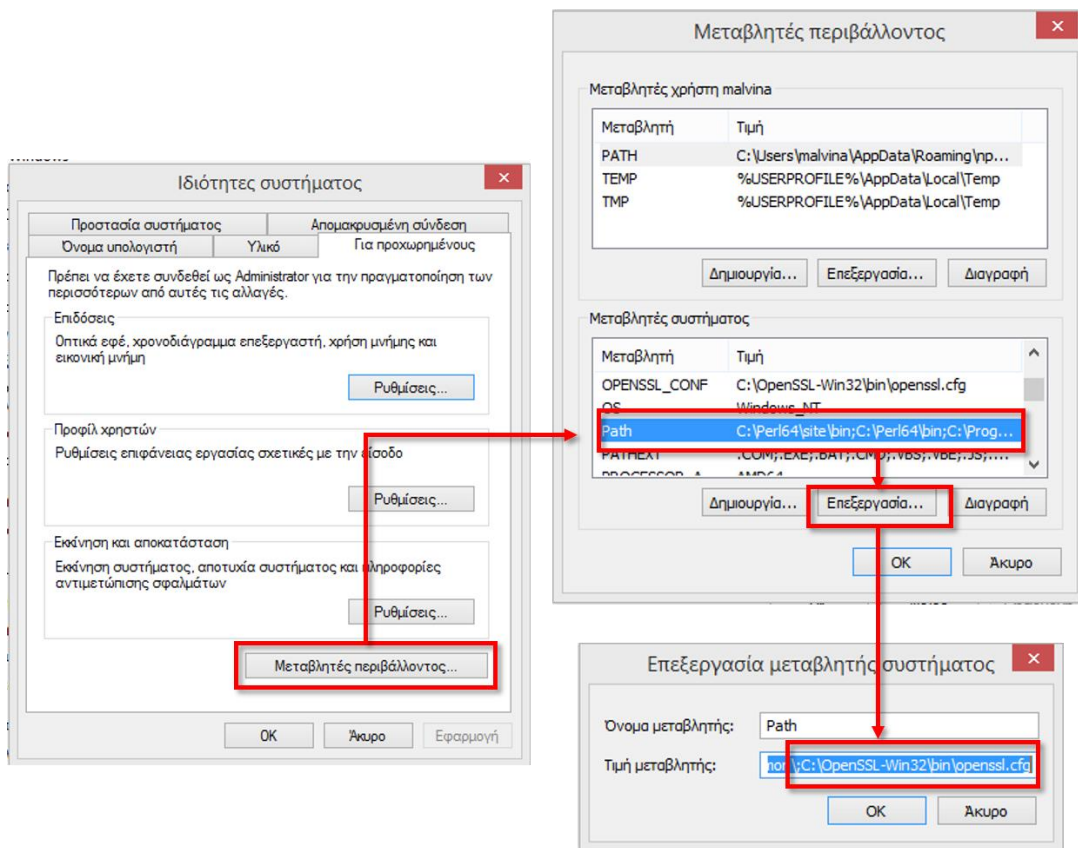
Appendix A - Fig. 5. Installing OpenSSL

After the successful installation of OpenSSL and the running of its relative executable file, the warning message depicted in Appendix A - Fig. 5 is shown.



Appendix A - Fig. 6. OpenSSL warning message

This warning is bypassed by modifying and adding OpenSSL to Path system variables.



Appendix A - Fig. 7. Modifying system's variable Path

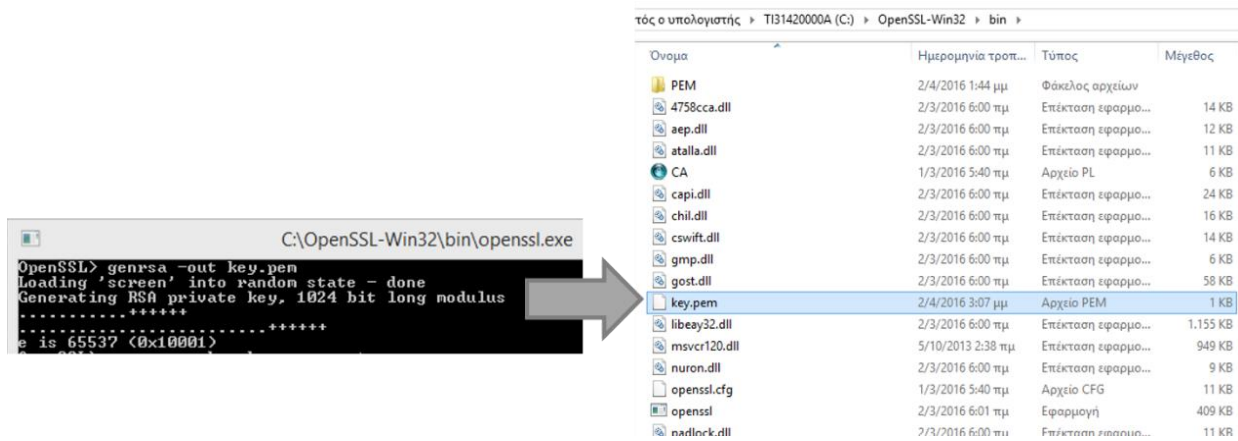
Generating self-signed certificates

Once the installation and path configuration have been completed, we can safely run OpenSSL and create our self-signed certificate following the steps mentioned adown.

Step 1: Run the OpenSSL.exe

Step 2: Generate a Private Key. This step involves the creation of the RSA Private Key and its storage in a PEM format in order to be readable as ASCII text.

➤ `openssl genrsa -out key.pem`



Appendix A - Fig. 8. Generating a Private Key

Step 3: Generate a CSR (Certificate Signing Request). Once the private key is successfully generated, a Certificate Signing Request has to be created. During this step, several prompts will appear asking for the below mentioned information to be filled in.

➤ `openssl req -new -key key.pem -out csr.pem`

Country Name (2 letter code) [GB]:
State or Province Name (full name) [Berkshire]:
Locality Name (eg, city) [Newbury]:
Organization Name (eg, company) [My Company Ltd]:
Organizational Unit Name (eg, section) []:
Common Name (eg, your name or your server's hostname) []: *
Email Address []:
Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:

* NOTE: In the second command, when prompted for "Common Name (i.e. your name or your server's hostname) []:", it is actually the domain name field and not a custom generated name. Not supplying the appropriate domain name will result in "domain mismatch" error.

Step 4: Generating a Self-Signed Certificate. At this point is feasible the generation of a self-signed certificate for environments that do not desire to obtain a CA-signed certificate. Even though that the generated certificate has the same effect with any other certificate out there, web browsers will notify their clients for potential authentication risk, since it has been signed by an unknown authority and cannot be fully trusted.

```
➤ openssl x509 -req -days 9999 -in csr.pem -signkey key.pem -out cert.pem
```

Programming the server

After the generation of the self-signed certificates, we are ready to write our server with JavaScript language.

From the below JavaScript code, we can see

- a few calls to modules (https,fs etc.)
- the management of the index ("/")
- the generation of socket.io events like new_client, message and video call
- the emit statement within the connection event. This statement emits the message sent from one client to all the other clients using the same web server. At this point we notice that there are two emit statements, the previous one and the socket.emit("message", message); that only emits the message received from a client to that respective client.

```
var https = require('https');
var fs = require('fs');
var ent = require('ent');
var options = {
  key: fs.readFileSync('key.pem'),
  cert: fs.readFileSync('cert.pem'),
  NPNNProtocols: ['http/2.0', 'spdy', 'http/1.1', 'http/1.0']
```

```

};

var myApp = https.createServer(options, handleRequest);
myApp.listen(443);
function handleRequest(request, response){
  console.log('Path Hit: ' + request.url);
  if(request.url === '/'){
    fs.readFile(__dirname + '/index.html', function (err, data) {
      if (err) {
        response.writeHead(500);
        return response.end('Error loading index.html');
      }
      response.writeHead(200);
      response.end(data);
    });
  }else{
    fs.readFile(__dirname + request.url, function (err, data) {
      if (err) {
        response.writeHead(500);
        return response.end('Error in loading page');
      }
      response.writeHead(200);
      response.end(data);
    });
  }
};

var io = require('socket.io')(myApp); // use socket.io
io.on('connection', function (socket, username) {
  socket.on('new_client', function(username) {
    username = ent.encode(username);
    socket.username = username;
    socket.broadcast.emit('new_client', username);
  });
  socket.on('message', function (message) {
    message = ent.encode(message);
    socket.broadcast.emit('message', {username: socket.username,
    message: message});
  });
  socket.on('video call', function(data){
    socket.broadcast.emit('video call', data);
  });
});
};

```

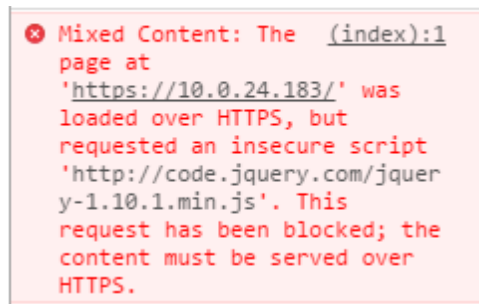
Appendix A - Table 1. Server code for conferencing

Integration of a client-side

After the successful running of Node.js web server a landing HTML page is presented to clients, which implements the connection with socket.io module. It is the most fundamental bidirectional communication step done in client-side since it emits messages from the client to a

web server, while the web server is responsible for the notification of the remaining clients using the same server. At this point, it is important to notice that if we request a file from an HTTP connection while we have HTTPS connection, the web browser runs with error for mixed content and the requested file blocked. An example of this error is presented in Appendix A - Fig. 9.

The best solution for this problem is to serve all content as HTTPS with fixing our links. Often, the HTTPS version of the content already exists and this just requires adding an "s" to links - http:// to https://, otherwise we can download and request the files from our folder.



Appendix A - Fig. 10. An example of a blocked mixed content in web console

The following code illustrates the html page for the client-side. As we can see, it is an HTML page with the basic HTML elements and also a JavaScript part of code. The JavaScript code communicates with the server's code. This communication is successful with the correct call of the socket.io. Two lines of code are responsible. One line in the head of the HTML page and one more in the JavaScript part.

The first one is the `<script src="/socket.io/socket.io.js"></script>` and the second one is `var socket = io.connect('https://10.0.24.183:443');`. The second line calls the server with the enabled port. If this address is false, the whole page runs with errors and no one client cannot communicate in real-time with other.

```
<!DOCTYPE html>
<html>
  <head>
    <script src="js/jquery-1.10.1.min.js"></script>
    <script src="/socket.io/socket.io.js"></script>
  </head>
  <body>
    <h1>Real-time Chat!</h1>
    Messages<br/><br/>
    <form action="/" method="post" id="chat_form">
      <input type="text" name="message" id="message" placeholder="Your
```

```

        message..." size="50" />
        <input type="submit" id="send_message" value="Send" />
    </form>
</section id="chat_zone"></section>

<script>
    var socket = io.connect('https://10.0.24.183:443');
    /***** chat with username and messages *****/
    var username = prompt('What\'s your username?');
    socket.emit('new_client', username);
    document.title = username + ' - ' + document.title;

    socket.on('message', function(data) {
        insertMessage(data.username, data.message)
    })

    socket.on('new_client', function(username) {
        $('#chat_zone').prepend('<p><em>' + username + ' has joined
        the chat!</em></p>');
    })

    $('#chat_form').submit(function () {
        var message = $('#message').val();
        socket.emit('message', message);
        insertMessage(username, message);
        $('#message').val('').focus();
        return false;
    });

    function insertMessage(username, message) {
        $('#chat_zone').prepend('<p><strong>' + username +
        '</strong>' + message + '</p>');
    }

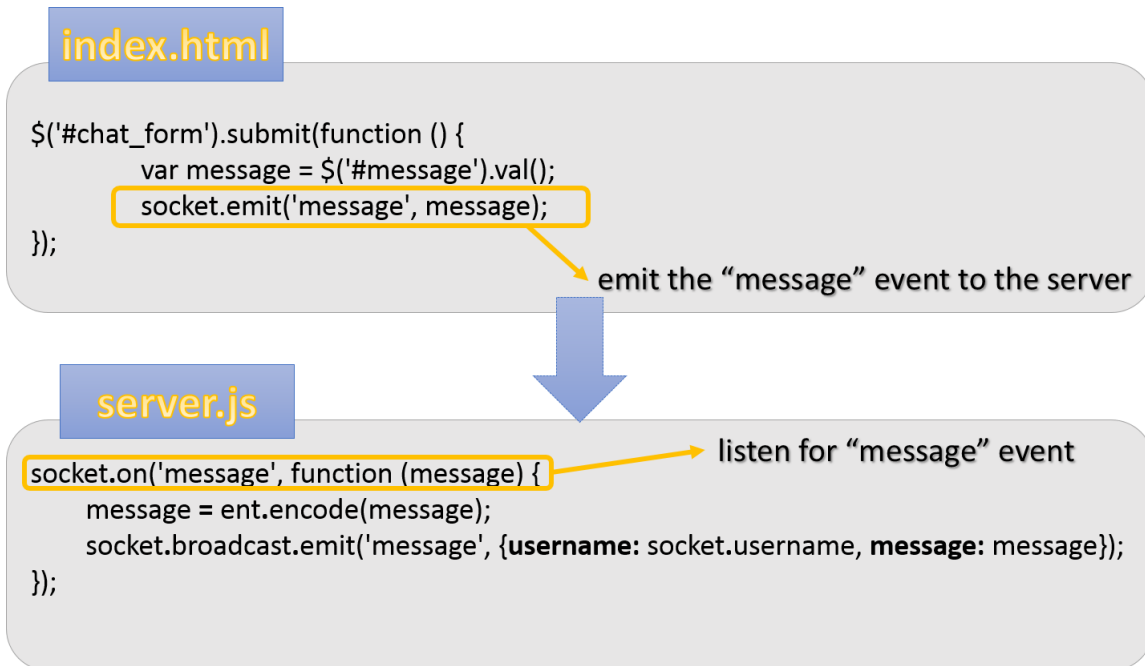
</script>
</body>
</html>

```

Appendix A - Table 2. Client's code for conferencing

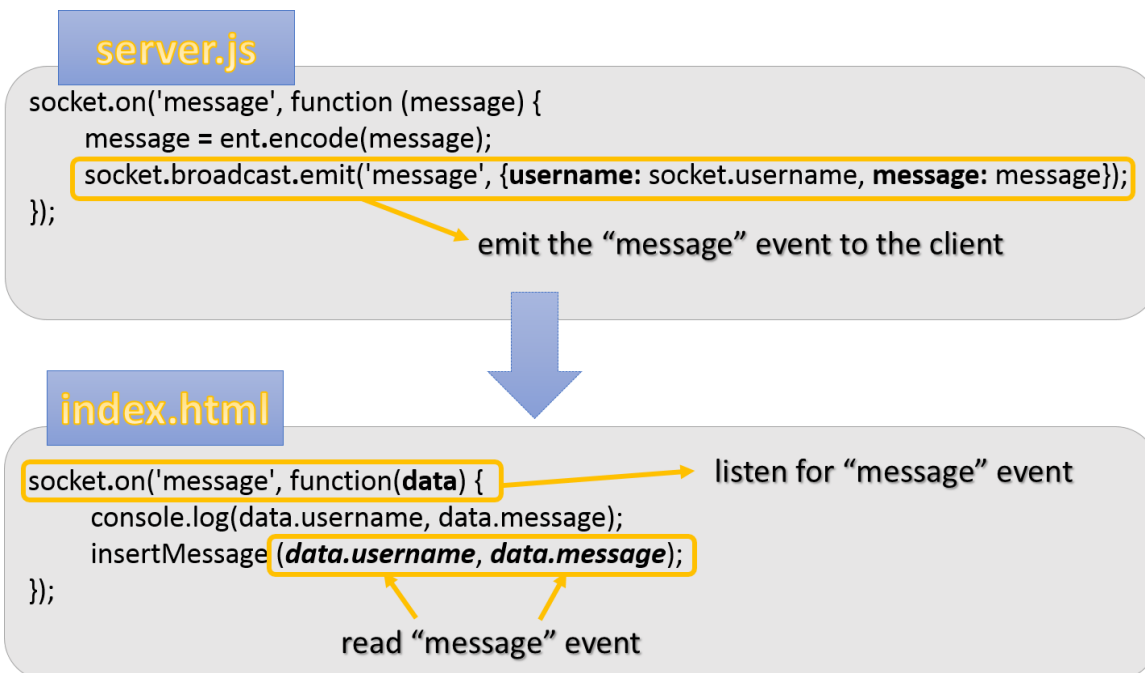
Appendix A - Fig. 9 and Appendix A - Fig. 10 demonstrate the process of a message from client to server and from server to other clients. The server and the client send each other events with `socket.emit()` and listen to the events that are sent with `socket.on()`.

The client wants to send a message to the server. When we submit the form, a message will be sent to the server with the emit function along with the content of the form message.



Appendix A - Fig. 11. Socket.io process from client to the server

When we do a `socket.emit()` on the server side, we can send *broadcast* the message. This means that the message is destined for all the other clients with the exception the client that sends the message to the server.



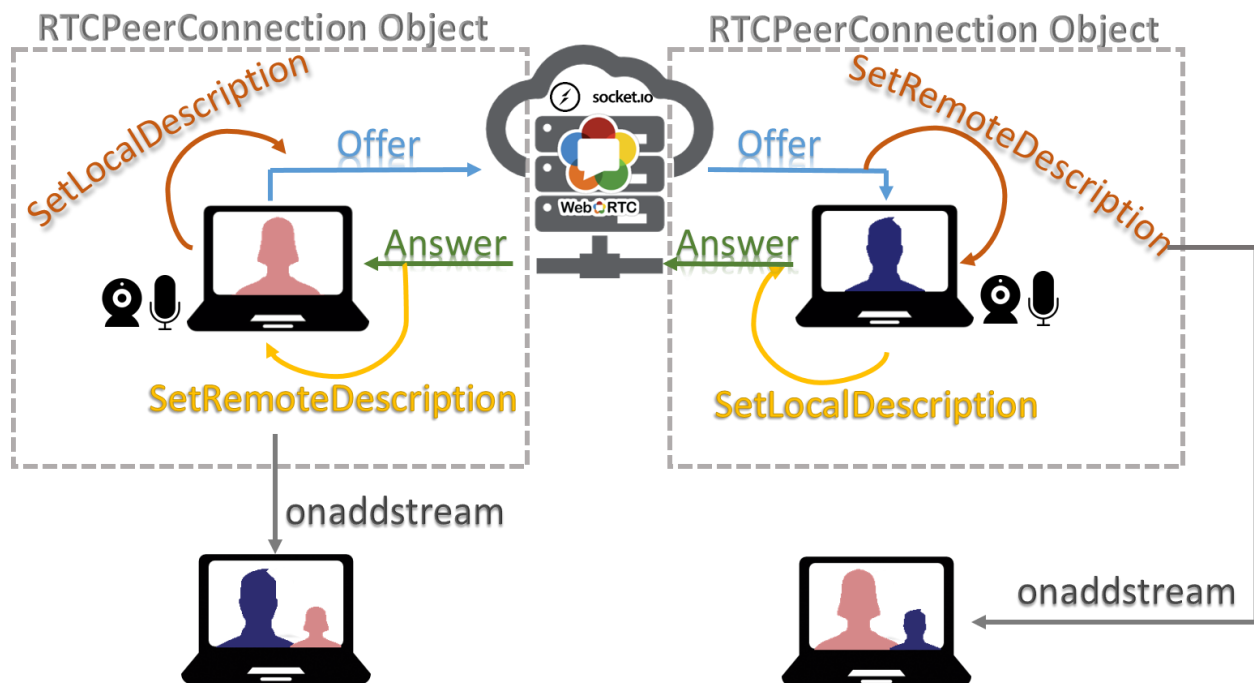
Appendix A - Fig. 12. Socket.io process from server to the clients

RTCPeerConnection for video calls

An RTCPeerConnection is responsible for the communication between two browsers/users and with the use of getUserMedia API has a set of two streams. The first stream is the local stream that become from the local microphone and webcam and the second stream is the stream that is received from the other user through this RTCPeerConnection.

Appendix A - Fig. 11 illustrates the process for the WebRTC call with Socket.io library as signaling server. The whole process has the following steps:

1. The first user sends the LocalDescription using the setLocalDescription with the CreateOffer() function to the second user
2. The second user sets the received description as RemoteDescription using the setRemoteDescription() and handles the stream with onaddstream to his webpage.
3. The second user runs the CreateAnswer() function with his LocalDescription
4. The first user receives the description of the second user, it sets as RemoteDescription and call onaddstream to add the remote stream to the webpage



Appendix A - Fig. 13. WebRTC video call process

Appendix A – Table 3 describes the functions that are used for the successful call. The RTCPeerConnection uses the below callback methods [47]

Method	Description
onicecandidate	It indicates the addition of a new candidate to the local peer by ICE protocol machine inside the browser
addIceCandidate	It adds a remote candidate to the ICE Agent remote description, checking at the same time for connectivity issues amongst other new candidates in case IceTransports constraint is not set to “none”
onaddstream	It is called each time a new MediaStream is added by the remote peer
onremovestream	It is called each time a MediaStream is removed by the remote peer
addStream	It adds a new stream to an RTCPeerConnection object
removeStream	It removes a stream from an RTCPeerConnection object
createOffer	The constraints parameter may be supplied to provide additional control over the offer generated. It generates an SDP blob offer specially configured for the current session and conforming with RFC3264 offer. It contains the description of the local MediaStreams, the browser’s supported RTP/RTCP/codec options and ICE Agent’s candidates. The supplement of constraints parameter is optional and can be used for better control over the generated offer
setLocalDescription	It notifies an RTCPeerConnection object to treat the supplied RTCSessionDescription as a local description
setRemoteDescription	It notifies an RTCPeerConnection object to treat the supplied RTCSessionDescription as a remote offer or answer
createAnswer	It generates a compatible SDP answer for the current session based on the parameters of the remote configuration

Appendix A - Table 3. WebRTC functions for video call

Appendix A - Table 4 demonstrates a small snippet of code is presented in the following table for the process of point-to-point video call.

```
socket.on("video call", function(data) {  
  
  switch(data.type){  
    case "iceCandidate":  
      pc.addIceCandidate(new RTCIceCandidate(data.candidate));  
      break;  
  
    case "offer":  
      pc.setRemoteDescription(new  
      SessionDescription(data.description), function() {  
        console.log("failure callback");  
      });  
      pc.createAnswer(function(description) {  
      pc.setLocalDescription(new  
SessionDescription(description);  
        socket.emit("video call", {type: "answer",  
          "description": description});  
      }, function() {console.log("failure callback");});  
      break;  
  
    case "answer":  
      pc.setRemoteDescription(new  
      SessionDescription(data.description), function() {  
        console.log("failure callback");  
      });  
      break;  
  }  
});
```

Appendix A - Table 4. Code for point-to-point video call with WebRTC and Socket.io

Appendix B - Glossary of Technical Terms

HTML5

HTML5 is the latest evolution of the HTML standard which is used as a representation mean for the content of webpages viewed into users' browsers. It is the latest revision that was published on 28 October 2014 by the W3 Consortium (W3C), enriched with several advanced webical technologies and features. Some of the most notable features is user's ability to "drag and drop" webpage elements, the support of a full-duplex communication protocol known as WebSockets, and the addition of canvas element and the streaming of audiovisual content without using any kind of plugins. Due to the wide variety and differential nature of these features, websites and applications tend to classify them into one or more of the following groups based on their functionalities: semantics that provide more precise description in a webpage, multimedia for audio and video, 2D or 3D graphics, connectivity for the communication with the server, device access for use with different devices, performance that provides speed optimizations and better hardware utilization, offline storage for saving data locally on the client-side

JavaScript

JavaScript is a dynamic, structured programming language which complies with ECMAScript language specification. It has robust capabilities that involve text scripting, regular expressions, arrays and dates. The execution of scripts written in JavaScript language is done with the assistance of a JavaScript engine, which is embedded in the environment hosting that script, usually a web application, browser or plugin. JavaScript language provides support for client and server side scripting on the Web. Concerning **client-side** behavior of HTML pages, JavaScript has the ability to load specific content of a page via AJAX without reloading the entire page, animate the page components (moving them, resizing them, draw them , etc), interact actively with content for gaming purposes, validate the input values of a web form before its postage to the server, receive information from the users for Web analytics.

As a result, client-side scripting provides advanced capabilities for the manipulation and dynamic loading of web pages' content. This type of scripting is usually met on browser applications where the asynchronous communication is a must.

On the other hand, **server-side** scripting involves operations performed by server in order to reduce the workload of client information. Netscape was the first browser that introduced the idea of server-side scripting back in 1995. Today, JavaScript comes with various classes, objects and functions, which extend the original language and provide server-side capabilities. The state-of-art implementations make use of Node.js [48], a special runtime environment that deals with database storage and retrieval with a few only scripts. Besides client and server side scripting, JavaScript language provides a couple of notable features like:

- ❖ JSON (JavaScript Object Notation), a generic data format that is characterized as a subset of the JavaScript object syntax.
- ❖ jQuery, a popular feature-rich and fast JavaScript library. It was designed to make easy the traversal of DOM trees, the handling of events, the animation of elements and others. jQuery remains the most widely used library in the majority of browsers, with several plugins authored for the likes of end-users.

Today, JavaScript is being used as a general purpose programming language from web-based implementations down to plain digital documents. It is also deemed to be the scripting language of World Wide Web, thanks to its dynamic cross-platform capabilities and lightweight requirements.

MySQL

MySQL is one of the most widely used open source relational database management systems (RDBMS), which has been adopted by the majority of web applications thanks to its reliability and reduced complexity. Beyond the state of art features in RDBMS, MySQL provides an unprecedented ease of use, native cross platform support and improved database scalability through MySQL Replication technology [49]. All these distinctive features make MySQL superior to other SQL solutions, providing mainly fast read and write operations over the data held in the respective database. This latest capability makes it the ideal database for every kind of application which demands high performance, regardless of the amount of data being processed. Based on these characteristics, the riskless procedures held from the database system and a vast number of users confirm daily its stability and data integrity, MySQL was clearly the best choice amongst the RDBMS offered through the Jena SDB triple store.

JSP

JSP is the technology that used to develop interactive web applications. A JSP page has the classic HTML format with embedded Java code by using the special JSP tags, which start with <% and end with %>. When a JSP page is called, it is compiled by the JSP engine into a Java servlet and this phase is well-known as translation. The servlet engine produces an executable class and forwards an output in HTML format inside an HTTP response and the web browser parses the dynamic HTML page inside to an HTTP response. The advantages of the JSP is the portability because it deployed in many platforms, the reusability of the servlet components and the simplification of the development process.

SERVLETS

The servlets support the programming model of the request and the response. As a result, when a user sends a request to the specified server, the server sends this request to the servlet. After that, the servlet sends the response to the server and the server sends back to the user. For each request, servlet specifies if a request is GET or POST operation. Depending on request, servlet calls doGet or doPost methods that takes request (HttpServletRequest) and response (HttpServletResponse). Generally, servlets are Java classes which produce dynamic HTML content. With the use of the servlets, we gain all the benefits from the Java platform.

Web Ontology Language (OWL)

The Web Ontology Language (OWL) is a language specially made for the Semantic Web, in order to sufficiently describe concepts stemming from a specific domain and organize them in a hierarchical manner. This organization takes place in such a way that primarily aids computer software to understand these concepts, rather than presenting this information to humans.

Jena Framework

Jena is a widely known framework based on Java language, which has been used for the development of various Semantic Web applications. It is an open source project that provides several Java APIs for dealing with OWL, RDF, RDFS, SPARQL, and other standardized languages or reasoning engines. It works as an additional layer that translates the semantics into

Java artifacts (classes, methods, attributes, etc.), supporting all three sublanguages (Full, Description Logic and Lite) of OWL and adding methods for the validation of ontologies via a fully functional OWL API.

Jena SDB

Jena SDB is a native triple store component of Jena framework for the persistent storage and querying of RDF data using third party databases. Triple stores are custom-build databases that provide scalability and query-rich functionalities over RDF datasets. SDB belongs to DBMS category, which means that it is taking advantage of the storing and retrieval mechanisms granted by the underlying database system of the application.

SPARQL (SPARQL Protocol and RDF Query Language)

SPARQL is a Semantic Web standard and the most widely used query language for RDF datasets. Its latest protocol provides improved performance and advanced retrieval capabilities, thanks to a set of unique features like SPARQL algebra, custom filter functions, aggregation, various storage systems support, etc.

X3D Standard

X3D standard comes with a long history of representing web-based 3D graphics, since it constitutes the successor of the legacy VRML (Virtual Reality Modeling Language). It makes use of the XML format. Nowadays, X3D is the most widely used standard for the presentation of 3D content on the Web, using efficient XML validation mechanisms for the connotation of scripting errors to content's authors.

X3DOM

X3DOM is able to incorporate 3D content on the Web without the installation of plugins, while at the same time, it provides an efficient integration mechanism for the current Web standards. Furthermore, its novelty lies to the merging of HTML5 and X3D standards through a viable framework, which is supported by the majority of today's browsers.

Web Real-Time Communications (WebRTC)

The Web Real-Time Communications (WebRTC) standard increases the multimedia capabilities of today's browsers by providing an API for the direct real time communication between two or more computers. This interaction takes place in users' web browser without the use of extra plugins, making easy to share data and multimedia (video and audio). WebRTC is a standard which was developed by the World Wide Web Consortium (W3C) and the Internet Engineering Task Force (IETF) as a free, open source project, that aims not only to enrich current browsers, but also the mobile applications with the functionalities provided by the Real-Time Communications APIs.

STUN

Session Traversal Utilities for NAT (STUN) is a client-server protocol. The server part discovers the user's IP address that is public and the TCP port. The STUN client be noticed for these. This give to the WebRTC peer the opportunity to get its public IP address and port, and passes them via WebRTC signaling to the other peer. STUN gives permission to the WebRTC media and data flows like real-time communication with voice, video, and messaging between peers.

MCU (Multipoint Control Unit or Multipoint Conference Unit)

The vigorous network topology for conference communications is the MCU (Multipoint Control Unit or Multipoint Conference Unit). MCU is used as a bridge. It is a server which is used when big number of participants want to connect to a single session and distribute multiplexed media content to each other. Whenever a participant (peer) decides to leave the specified conference, the MCU simply terminates this connection. A MCU can handle different codecs and resolutions to enable the interoperability between the participants.

Node.js

Node.js is a cross-platform environment especially made for the development of fast, scalable network applications. It started and remained as an open-source initiative for server-side web applications, using a non-blocking I/O model to optimize scalability in real-time applications. Node.js is packed with npm ecosystem, the largest open source library repository that provides advanced asynchronous and lifecycle control capabilities at runtime environment.

Socket.io

Socket.io is used for the implementation of real-time applications in Web browsers and mobile devices, enabling synchronized communication mechanisms between multiple users with just a few lines of JavaScript code.

HTTPS

HTTPS is the HTTPS protocol over TLS/SSL which is responsible for exchange of data between clients' browser and a website. The 'S' at the end of HTTPS stands for 'Secure', denoting that within this protocol a connection is encrypted by Transport Layer Security or Secure Sockets Layer.