FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

CIDESI®

Design and Development of a 3D Dynamic Configuration Interface for an Industry 4.0 Assembly Cell

A Thesis

Submitted to the Faculty of Fachhochschule Aachen and Centro de Ingeniería y Desarrollo Industrial

BY

José Gerardo Gómez Méndez

In Partial Fulfillment of the requirements
for the degree of Master of Science in Mechatronics

Santiago de Queretaro, Qro., Mexico, February 2018

Declaration

I hereby declare that this thesis work has been conducted entirely on my own accord.

The data, the software employed and the information used have all been utilized in complete agreement to the copyright rules of concerned establishments.

Any reproduction of this report or the data and research results contained in it, either electronically or in publishing, may only be performed with prior sanction of the University of Applied Sciences (FH Aachen), Centro de Ingeniería y Desarrollo Industrial (CIDESI) and myself, the author.

José Gerardo Gómez Méndez

Santiago de Queretaro, Qro., Mexico, February 2018

Acknowledgments

Firstly, I would like to express my immense gratitude to Conacyt for the opportunity to have a scholarship and for supporting the study of my master’s degree in Queretaro, Mexico, and Aachen, Germany.

I would also like to thank my thesis advisor in Germany, Prof. Dr.-Ing. Jörg Wollert of the FH Aachen. He steered me in the right direction whenever he considered necessary, and he guided me during the development of this thesis. Furthermore, my sincerest gratitude to my second thesis advisor in Germany Prof. Dr. rer. nat. Klaus-Peter Kämper and all the members of the Embedded Systems Laboratory at FH Aachen for their support on the development of this project.

I would also like to thank my thesis advisor in Mexico Dr. Antonio Estrada Torres at CIDESI. I am grateful to him for his remarkably valuable comments and helpful advice on this thesis. In addition to Dr. Salvador Francisco Acuña Guzmán, Ing. Bertha Elisa Velasco Sánchez, Lic. Cruz Alicia Márquez Tecua and all the academic and administrative members at CIDESI.

I would also like to express my most profound gratitude to my parents, my grandparents, and the rest of my beloved family. Their support, knowledge, generosity, encouragement, and unconditional love during my master’s degree and my life.

Additionally, I would like to express my sincerest appreciation to my friends, colleagues, and family in Queretaro and Aachen, Jaime Robles, Osvaldo Picazo, Luis Rodríguez, Ricardo Vega, and Adrián Lizama, without their constant support, advisement, and genuine friendship, the accomplishment of this degree would not be possible.

Finally, I would like to thank my girlfriend Miriam Knöpfle for her help and encouragement throughout the writing of this thesis.

This master's thesis is dedicated

to my beloved mother Ana Laura Méndez Maín,

my cherished father Gerardo Gómez Salas,

and my adored grandparents Daniel Gómez Montano, María Luisa Salas Aguilar, Carlos Méndez Lara & Concepción Amelia Maín Morales.

Everything I am is thanks to you…

Terminology and Abbreviations

API	Application Programming Interface
GPU	Graphic Processing Unit
I4.0	Industry 4.0
IoT	Internet of Things
IP	Internet Protocol
JS	JavaScript
JSON	JavaScript Object Notation
LAN	Local Area Network
MB	Megabyte
RAM	Random Access Memory
REST	Representational State Transfer
UML	Unified Modeling Language

Abstract

The purpose of this thesis titled "Design and Development of a 3D Dynamic Configuration Interface for an Industry 4.0 Assembly Cell" is to present the design and development of a plugin for an existing e-commerce platform that is running inside an Industry 4.0 system for the construction of Lego Duplo cars.

The plugin provides the possibility of the adjustment of a product to the user's specific needs with the help of a detailed 3D preview of the output. Apart from being just a preview, it additionally allows a customization by predefined configuration parameters defined by the administrator. This feature intends to increase the intuitiveness of the application and therefore the satisfaction of the customer by providing a better experience in the purchasing process.

It also aims to produce high-quality software that is easily maintainable, well documented, bug-free and scalable to face any other challenges like displaying any type of product that is built using Lego Duplo bricks.

Resumen

El objetivo de esta tesis titulada "Diseño y desarrollo de una interfaz dinámica de configuración 3D para una celda de manufactura para la Industria 4.0" es presentar el diseño y desarrollo de un complemento para una plataforma de comercio electrónico que actualmente está en operación en un sistema de Industria 4.0 para el ensamble de automóviles con piezas de Lego Duplo.

El "plugin" provee la posibilidad de ajustar un producto a las necesidades específicas del usuario con la ayuda de una vista previa detallada en 3D. También permite la personalización del producto, basado en parámetros de configuración predefinidos establecidos por el administrador. Ésta característica tiende a incrementar la intuición de la aplicación y por lo tanto la satisfacción del cliente, al proporcionar una mejor experiencia en el proceso de compra.

También tiene como objetivo producir software de alta calidad que sea fácilmente mantenible, bien documentado, libre de errores y escalable para afrontar los futuros retos como mostrar cualquier tipo de producto que sea construido usando ladrillos de Lego Duplo.

Kurzfassung

Die Zielsetzung dieser Masterarbeit unter dem Namen „Design und Entwicklung einer Dynamischen 3D Konfigurationsschnittstelle für eine Industrie 4.0 Montagezelle“ ist es, ein Plugin für eine bereits bestehende E-Commerce Plattform zu kreieren, die in einer Industrie 4.0 Plattform zur Konstruktion von Lego Duplo Fahrzeugen läuft.

Das Plugin bietet dem Benutzer mithilfe einer detaillierten 3D-Vorschau die Möglichkeit, ein Produkt nach seinen eigenen Anforderungen anzupassen. Neben der Vorschaufunktion bietet es außerdem die Möglichkeit der Personalisierung mit vom Administrator vordefinierten Konfigurationsparametern. Diese Funktion beabsichtigt die Steigerung der Intuitivität der Applikation und damit der Kundenzufriedenheit, indem ein besseres Kauferlebnis ermöglicht wird.

Des Weiteren soll eine einfach zu pflegende, fehlerlose Software von hoher Qualität mit guter Dokumentation und einfach zu realisierenden Skalierungsmöglichkeiten geschaffen werden, um auch andere Herausforderungen wie das Anzeigen anderer Produkttypen mit Lego Duplo Bricks meistern zu können.

Contents

List of Figures

List of Tables

Introduction

Progression and changes in the way products are created have been a part of the evolution of industry since the creation of the first steam engines in the 18th century. There have been three major changes in the way of how industry is conceptualized, labeled as Industrial Revolutions. The last notable change was formalized in the mid-1970s where electronics and IT solutions began to expand quickly into the industry [1].

The past decade, the involvement of IT systems and electronics in the industrial sector has not only been widely adopted, but nowadays is an essential element. Processes are becoming more flexible with the integration of new trends like the Internet of Things (IoT).

This combination with IoT and the notable increase in the flexibility of IT systems resulted in the German government proposing a strategic initiative called Industry 4.0 that was adopted as a part of the "High-Tech Strategy 2020 Action Plan" [2][3].

The aforementioned necessity for the conception of a significant change in the industry is a result of the constant changes of the customer needs. Producing a high degree of product individualization, increasing digitalization and system integration and effective and efficient manufacturing operations are just common requirements nowadays that have in combination the aim of reaching high quality at low cost as well as the well-being of employees [4].

Theoretical Background

Industry 4.0

Industry 4.0 (I4.0) is not a mere vision on how the final products should be, or how the work environment should be upgraded. It presents a total integration of the manufacturing process. This integration does not only involve the players in the industry sector, but the whole supply chain and external stakeholders. Therefore, it implements three kinds of integration that are described in the following paragraphs [2].

The first type of integration is a horizontal integration through value networks that enables the collaboration with suppliers and customers [2].

The second type is the vertical integration of hierarchical subsystems inside a factory that produces flexible and reconfigurable manufacturing systems [2].

The last type of integration represents an “End-To-End Engineering Integration” in the product-centric value creation process [2]. This integration has a primary intention of creating value for customers through the involvement in the process from the beginning [7].

Another important aspect that is mentioned by many authors is the involvement of the user with the industry and there are some standards to achieve this. The ISO 9241-210:2010 in Part 210 for example establishes some paradigms for human-centered design for interactive systems where the users are involved throughout the design and development phase of a product.

Moreover, the vertical integration aims to support customized consumer demand. In order to do that it is necessary to produce small-lots and therefore to transfer the traditional factory into a highly flexible and reconfigurable manufacturing that leads to the implementation of a so-called smart factory [2].

Smart Factory

Smart Factory is a production solution that intends to meet today’s market needs in a flexibly and efficiently way by constituting a key feature of Industry 4.0 [7]. Smart factories are capable of managing complexity, are less prone to disruption and can manufacture goods more efficiently [3].

A Smart Factory achieves a higher efficiency in the production by the union of technology in every step of the process starting with the selection of raw materials, passing through the manufacturing process and even after the delivery to the user. By implementing cyber-physical production systems the management can be provided with more detailed real time information which presents one of the fundamental principles of a new era in the sphere of industrial production [7].

Internet of Things

The IoT provides the mean to achieve extremely flexible, individualized and resource friendly mass production. The basic concept of IoT is that products within a manufacturing process should find their way independently throughout the production process in intelligent factories by making products and devices work cooperatively and interconnected [1].

At its essence, involving Industry 4.0 and the Internet of Things is portrayed by the incorporation of physical and digital components to manufacture new products [8]. Through unique addressing schemes, essentially on the internet, objects interact and cooperate with each other to reach common goals [9][10]. These goals can be very diverse and reach from solving daily tedious tasks, to changing the face of medicine or to helping industrial manufacturing to be more efficient.

Problem Statement

The Industry 4.0 Lab at the FH Aachen provides the tools and mechanisms to interconnect different assembly cells, so synergistically, they carry out the assembling process of customized Lego Duplo cars.

The current area of opportunity in this system is the customization process. In Industry 4.0, it is essential to support customized demands by the customer; hence the involvement of the final customer with the assembly process is fundamental. Nonetheless, to let the user customize its own products so they can be suitable to build, the infrastructure of the system should provide the necessary means to customize the outcome within the restrictions of the assembling process.

Furthermore, another critical point in the side of Industry 4.0 is the scalability and flexibility that any system should provide, this delivers a considerable responsibility when developing software for this purpose. The only method to provide a system with the necessary means to be scalable resides in a correct use of the Software's Engineering.

This last point always represents a challenge to the designers of the program because not only the software has to work well; additionally, it has to be completely reliable, bug-free and with meaningful documentation that makes it possible for future engineers to understand the code and adapt it to the new functionalities that will attain eventually.

Moreover, the usage of IT tools such as PCs, tablets, and smartphones opens an area of opportunity for the interaction of the user with the system. Most devices used nowadays possess enough characteristics to support features like 3D visualization, rendering of complex images, and high bandwidth connectivity. All these points make the integration of the upcoming technologies with any process inside Industry 4.0 feasible, but also crucial.

Justification

This thesis attempts to integrate the customization process inside major IT devices, specially oriented to the touch-displays present in tablets and smartphones.

It is projected to use some of the techniques for the good practice of software development. One of these practices is the delivery of vast documentation, mostly based in UML 2.0 that makes the system completely clear to understand. The second point is the requirement of reliability, taking a considerable attention in the testing process, so the system is examined against all possible failures. This aspect is described in the section Results, so the future administrator of the system knows contingency methods against some of the crashes that may occur.

The last point resides in the flexibility. This project aims to have a flexible and scalable system; therefore, it provides the tools to adapt it to distinct requirements, making it possible to adjust it to the latest technologies and giving the plausibility to use it for any purpose different than Lego Duplo cars as long as it is build using Lego Duplo bricks.

Hypothesis

It is possible to design and develop a 3D web application for the customization of predefined Lego Duplo cars. Additionally, it is plausible to implement it inside an already operating Industry 4.0 system

Objective

To design and develop a 3D web application for the customization of predefined Lego Duplo cars, and to implement it inside an already operating Industry 4.0 system.

Requirements

1. To explore different platforms for the development of portable, scalable and robust applications.
2. To create an application that can receive the template of any Lego Duplo car template as an input parameter and provide the tools for customizing it and transmit the modifications back to the sender.

3. To develop an intermediate program that interprets the data from the database where all the templates are stored and translate this data into a standard format understandable by the Viewer (JSON).
4. To adapt both programs inside an OpenCart plugin, so that they can be integrated into the running Industry 4.0 platform.
5. To provide all the documentation for the installation and maintenance of the system.

Methodology

The adopted methodology on the development of the software was the "Incremental Process Model". The important on choosing a specific process model is because "it provides stability, control, and organization to an activity that can, if left uncontrolled, become quite chaotic [5]". According to R. Pressman [5]:

> "…when an incremental model is used, the first increment is often a core product. That is, basic requirements are addressed, but many supplementary features (some known, others unknown) remain undelivered. The core product is used by the customer (or undergoes detailed evaluation). As a result of use and/ or evaluation, a plan is developed for the next increment. The plan addresses the modification of the core product to better meet the needs of the customer and the delivery of additional features and functionality. This process is repeated following the delivery of each increment, until the complete product is produced… [5]".

The incremental model applies linear sequences in a staggered fashion as calendar time progresses. Each linear sequence produces deliverable "increments" of the software [6]. Figure 1 presents an example of this process model in a graph of project calendar time against the software functionality and features.

The main advantage in this process model is that all the features, functionalities and even the scope of the system can be refined and adapted to the changes in the other systems. Regarding the quality of the software, the possibility of testing the software in each iteration of development reduces the number of bugs considerably and prevents the appearance of significant errors that can take most of the developing time.

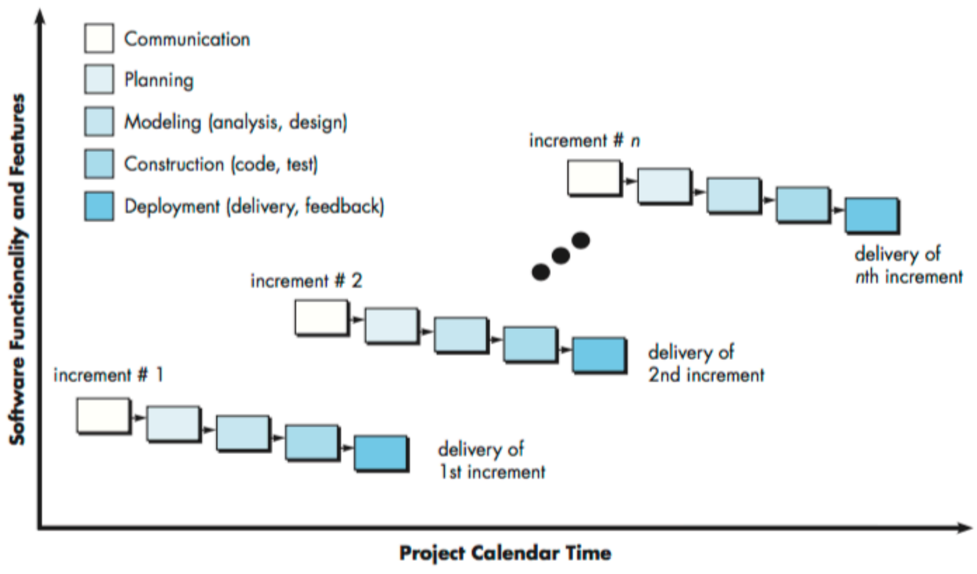

Communication
Planning
Modeling (analysis, design)
Construction (code, test)
Deployment (delivery, feedback)
increment # 1
delivery of 1st increment
increment # 2
delivery of 2nd increment
increment # n
delivery of nth increment
Software Functionality and Features
Project Calendar Time

Figure 1 Incremental Process Model, calendar time progress (Retrieved from R. Pressman [5]

Requirements Specification

Hardware Requirements

The requirements in terms of hardware to run the 3D Viewer consist in any computer with a shader-capable GPU. The needed peripherals can be either a multi-touch-screen or a screen with a mouse for the interaction. The system is also optimized to work on any tablet with at least 512 MB of RAM.

Software Requirements

The Viewer is designed to be used on web browsers. Any web browser capable of running WebGL 2.0 is able to display the system and basically any operative system can be used as well.

Software Used

OpenCart

"OpenCart is an open source PHP-based online shopping cart system. A robust e-commerce solution for Internet merchants with the ability to create their own online business and participate in e-commerce at a minimal cost. OpenCart is designed feature rich, easy to use, search engine friendly and with a visually appealing interface [11]".

This e-commerce platform provides several capabilities both for the user and for the administrator and it grants a simple store setup, performance and usability, since it uses AJAX technology in order to reduce load time. Also, its admin panel is simple to use but also provides the possibility to use numerous features to increase usability for clients: one-page checkout, smart site structure that simplifies shop navigation or handy search [12].

Furthermore, it is possible to customize OpenCart through a variety of software, modules, add-ons and plug-ins available nowadays [13]. One of the main features of these add-ons is that the e-commerce solution lets any user create personalized plugins, based on the requirements that the application faces.

This previous feature is done via the OpenCart modification system which allows store owners to extend and edit the system functionality without directly modifying core files which can be translated in hard-to-debug errors. The modification system works with XML modification files formatted in a specific way. These XML modification files give information to the system about which file should be changed, which code should be changed and what changes are to be performed on this code [14].

Apart from this XML modification file, it is possible to include an upload folder containing any amount of data necessary to make the add-on work; it is installed by compressing everything in a ".zip" folder with a ".ocmod.zip" extension [14].

Finally, the modification can be disabled or deleted from OpenCart in the "Modifications Panel", reversing all changes and thus the operation can be restored without any consequences.

OpenGL and WebGL

To describe what WebGL is, it is relevant to define another concept first. OpenGL is the premier environment for developing portable, interactive 2D and 3D graphics applications [15]; according to M. Woo and O. A. R. Board [16], it is essentially a software interface to graphics software.

> "OpenGL fosters innovation and speeds application development by incorporating a broad set of rendering, texture mapping, special effects, and other powerful visualization functions. Developers can leverage the power of OpenGL across all

> popular desktop and workstation platforms, ensuring wide application deployment [15].”

One of the main advantages of using OpenGL for displaying graphics is that it is designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms [16].

Now, speaking about WebGL, according to its developer group Kronos, is almost the same as OpenGL, but for the web. The original description given by them is as follows:

> “WebGL is a cross-platform, royalty-free web standard for a low-level 3D graphics API based on OpenGL ES, exposed to ECMAScript via the HTML5 Canvas element. Developers familiar with OpenGL ES 2.0 will recognize WebGL as a Shader-based API using GLSL, with constructs that are semantically similar to those of the underlying OpenGL ES API. It stays very close to the OpenGL ES specification, with some concessions made for what developers expect out of memory-managed languages such as JavaScript. WebGL 1.0 exposes the OpenGL ES 2.0 feature set; WebGL 2.0 exposes the OpenGL ES 3.0 API. WebGL brings plugin-free 3D to the web, implemented right into the browser. Major browser vendors Apple (Safari), Google (Chrome), Microsoft (Edge), and Mozilla (Firefox) are members of the WebGL Working Group [17]”.

How the previous definition can be summarized is that WebGL provides the same features as OpenGL, such as a cross-platform system that additionally is used in web browsers, which makes it even more accessible and portable.

Another highlight in using it according to T. Parisi [18], is that the developers can harness the full power of the computer's graphics rendering hardware using only JavaScript, decreasing the learning curve for new users with this approach.

Finally, the central aspect that should be highlighted is the way the coordinates are structured. The drawing takes place in a 3D coordinate system, including the depth [18]; therefore, it enables the user to locate 3D objects with different perspective views, all inside a canvas element.

PlayCanvas

PlayCanvas is a popular 3D WebGL Game Engine [19] that has the following characteristics according to Michael Larabel [20]:

> "The PlayCanvas Engine is a JavaScript library for building video games and includes graphics, physics, animation, audio engine, input device support (gamepads included), and an entity-component system. The WebGL-based graphics with this engine supports model loading, per-pixel lighting, shadow mapping, and post effects."

All the previous features and some others are summarized in the GitHub project [21], which also includes a full integration with 3D rigid-body physics engine and built-in components: model, sound, animation, camera, collision, light, rigidbody, script, particlesystem and different input peripherals like mouse, keyboard, touchpad, gamepad and virtual reality options are available. [21].

StarUML

StarUML is a sophisticated software modeler, and according to the web page "Methods & Tools":

> "StarUML is an open source software modeling tool that supports UML (Unified Modeling Language). It is based on UML version 1.4, provides eleven different types of diagrams, and it accepts UML 2.0 notation. It actively supports the MDA (Model Driven Architecture) approach by supporting the UML profile concept and allowing to generate code for multiple languages [22]."

Among the different characteristics it provides, StarUML supports major programming languages including Java, C#, and C++. You can generate source codes from your models or build a model from source code by reverse engineering. It also has the option to publish HTML Docs that can be viewed with most of the web browsers. All this previous information was retrieved from the official web page [23].

Autodesk FBX® 2013.3 Converter

The type of file that PlayCanvas requires for the CAD models is the "adaptable file format for 3D animation software" Autodesk FBX® which is a "3D asset exchange format that facilitates higher-fidelity data exchange between 3ds Max, Maya, MotionBuilder, Mudbox and other proprietary and third-party software [24].”

However, most CAD software like SolidWorks do not provide an export tool for this format; therefore, it was needed a software that converts a CAD file to this format. This software is Autodesk FBX® 2013.3 Converter that according to Autodesk:

> “Transfer files from one file format to another quickly and easily with the FBX Converter. This utility enables you to convert OBJ, DXF™, DAE, and 3DS files to or from multiple versions of the FBX format. New tools are now available with the FBX Converter 2012.1. You can view FBX animation files in real time with the FBX Viewer, explore and compare FBX file contents with the FBX Explorer, and manage animation takes with the FBX Take Manager [25].”

System Analysis

The implementation of the 3D Viewer as an OpenCart plugin divides itself into 4 systems, each one in charge of a specific task. The use case model presented in Figure 2 shows the functionality of the whole application and how the interaction among the subsystems works. System 1 corresponds to the implementation that was already running with the e-commerce platform OpenCart for the customization of the cars; in this case, this modification to the previous implementation intends to solve the requirement 4 in Section Requirements, and the description of the cases for this system is shown in Table 1.

The system 2 refers to the requirement 3 in Section Requirements, and is the primary communication among all the systems; just as before, the description of the cases is detailed in Table 2.

Subsequent, system 3 corresponds to the Requirement 2 in Section Requirements. This system is the one in charge of the primary functionality of this thesis; it provides the ability to modify a Lego Duplo car in a 3D environment. The description of the cases of this system is located in Table 3.

Finally, system 4 is just a vague representation of the “IoT Adapter” that was already operating. The only use case shown here is the one that was taken into functionality for the operation. However, this system provides much more functionalities, and here it is important to establish shared communication protocols and common rules for the notation of the items. The description of this particular case is in Table 4.

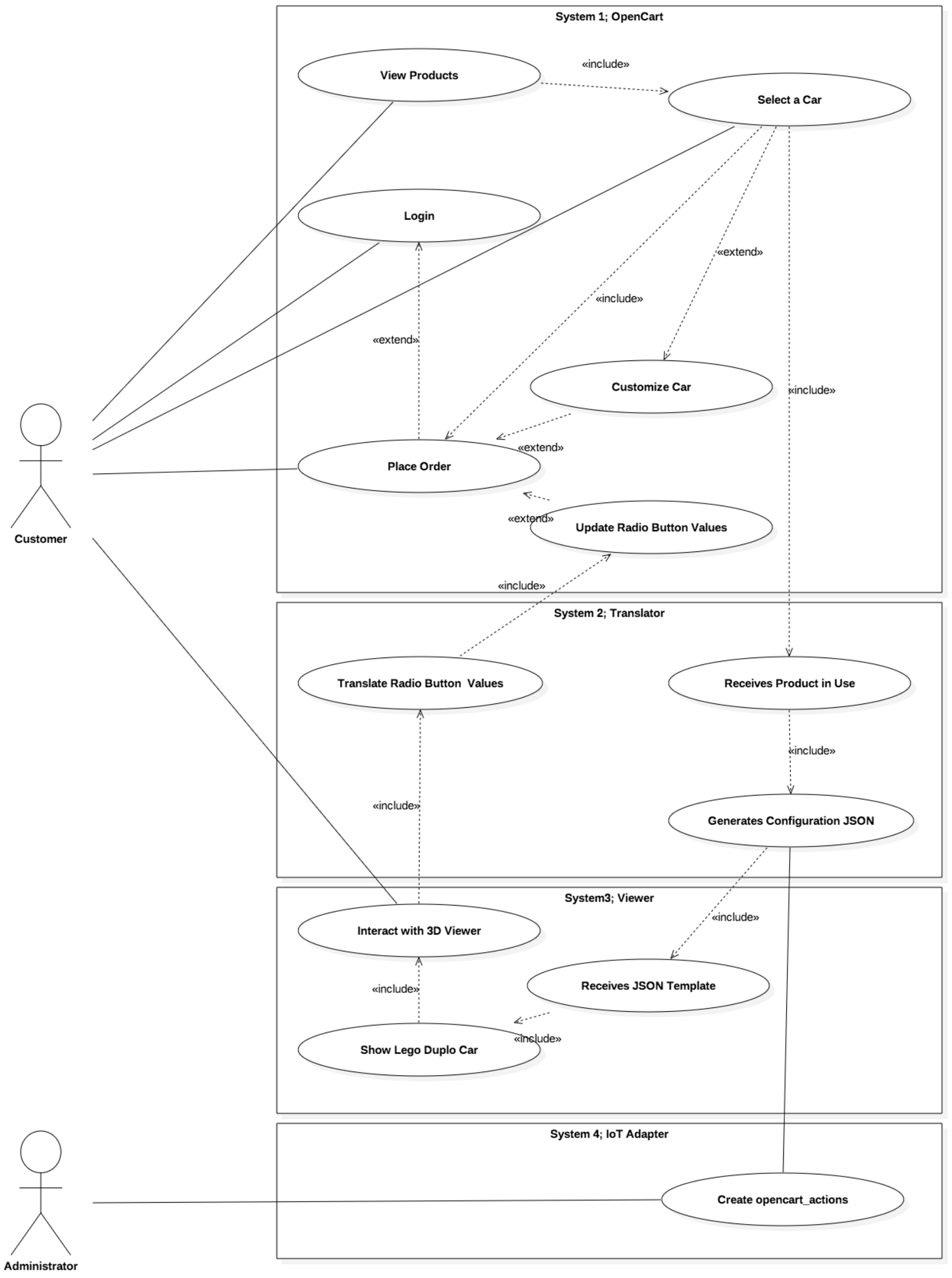


Figure 2 General Use Case for the system based on the Requirements

Use Case	Description
View Products	View all the products (Lego Duplo car templates).
Select a Car	Choose one of the previous products and open it for customization before the purchasing process.
Customize Car	The process of selecting the desired elements on the car, e.g. part colors or additional equipment. This process is done by the activation of radio buttons.
Place Order	Once a product is configured, the system creates an order with the specifications.
Login	The user can login to add the order to his record. This process is optional.
Update Radio Button Values	When the changes are made on the 3D Viewer, the system communicates with OpenCart and update the values of the radio buttons (they are hidden), so it is possible to place an order.

Table 1 Use Case Description for System 1

Use Case	Description
Translate Radio Button Values	The system gets the values of the active bricks and assigns them as the radio button value in OpenCart.
Receives Product in Use	When a car product is selected, it gets the information of the ID of the product, so that the template can be generated.
Generates Configuration JSON	Using the ID of the product, the system looks into the open_cart_actions table of the database and generates a JSON that serves as an input for the Viewer to display the product in use.

Table 2 Use Case Description for System 2

Use Case	Description
Receives JSON Template	The system receives the configuration template from System 2 and uses it to display the car.
Show Lego Duplo Car	Shows the basic template of the car in use, so that the user can customize it.
Interact with 3D Viewer	This case refers to the physical interaction of the user with the 3D Viewer. It can visualize an object and change the configuration of the car, using either a touchscreen or a mouse.

Table 3 Use Case Description for System 3

Use Case	Description
Create opencart_actions	When a new template is created, the administrator creates the building instructions for the specific template. In System 2 this is used to generate the configuration JSON.

Table 4 Use Case Description for System 4

System Design and Development

As described in the previous section, the whole customization program divides itself into four systems. In this segment, there is a description of the development and operation of each of them.

For future reference in this subsection, when some text is contained inside quotation marks it means is either a path, an extension, a module name, the text inside a button, or the name of a file (e.g., "/3DViewe.ocmod.zip") and if the text is formatted with italics is referring either an extract of the code (e.g., *function()*) or the name of a variable.

System 1 - OpenCart

As mentioned in Chapter 2, the e-commerce platform OpenCart is used to handle the orders from the users. Despite the features of this program, the standard customization for a product is based on a static image of the product plus some radio buttons to add or take away components, in this case for a Lego Duplo car as seen in Figure 3. This kind of customization presents several limitations, primarily because the customer is not able to perceive the final product. Additionally, to get a basic idea on the different perspectives of the product, it is necessary to add many images to present views from every angle, making it tedious to the administrator each time a new stock is added.

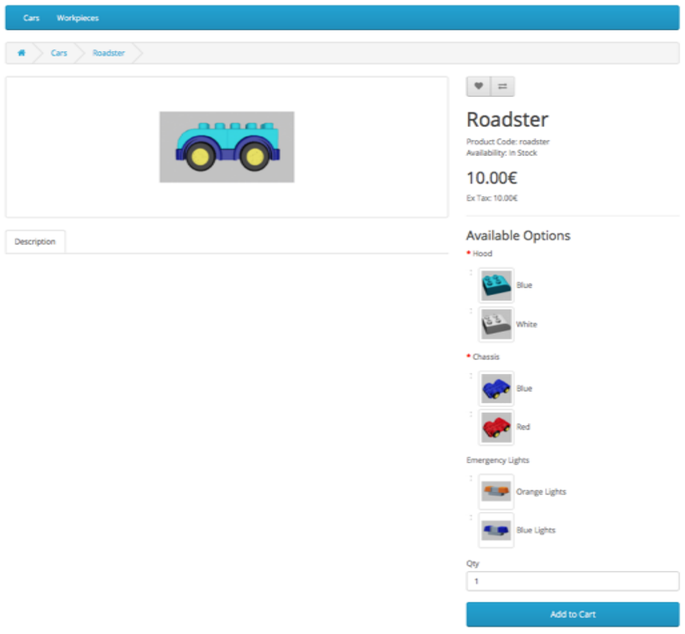
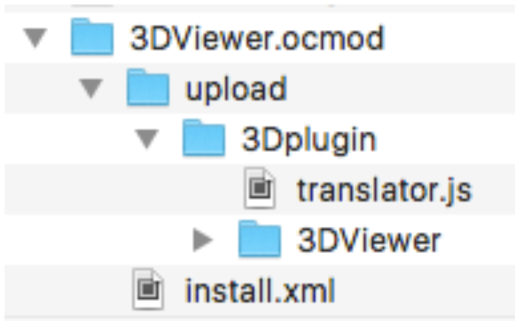


Figure 3 Common implementation of OpenCart

To enhance the action of selecting the desired elements on the order, a modification running as a plugin is created to alter the interaction with the user.

One of the main advantages in OpenCart is that instead of letting the user alter the source code of the program, it allows creating a .xml file with all the changes to be made. This point presents many features especially in the development process, but mainly when the administrator requires disabling a specific plugin.

To install the created modification on the admin panel of OpenCart, there is a section “Extension Installer”, where the “3DViewer.ocmod.zip” is uploaded. This “.zip” file contains the configuration files to add the 3D Viewer as a modification, as described in Figure 4.



```
▼ 3DViewer.ocmod
    ▼ upload
        ▼ 3Dplugin
              translator.js
          ▶ 3DViewer
      install.xml
```

Figure 4 Content of the folder “3DViewer.ocmod”

The only relevant file for "System 1 - OpenCart" is "install.xml". Inside, there are all the modifications to execute to the source code of OpenCart. To do that it describes first the path of a specific file, for example:

<file path="catalog/view/theme/default/template/product/product.tpl">

Then, it defines an operation to do to the file, for example:

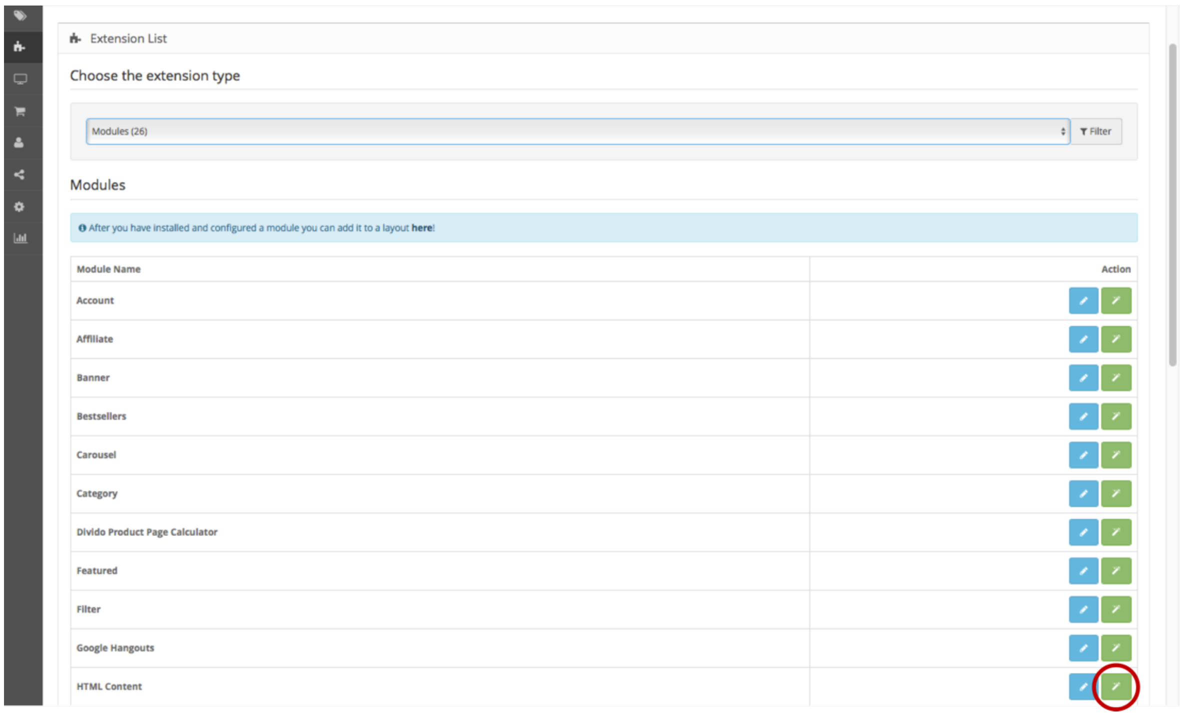
<search> <![CDATA[<?php echo $content_top; ?>]]></search>

With the preceding operation it is looking inside the code for the line or lines that contains *<?php echo $content_top; ?>*. After it finds the parameters, it describes a detailed action to do, in this case:

```
<add position="replace"><![CDATA[
    <?php
        if (strpos($content_top, '<h2>3DViewer</h2>'))
            $hide_pc=FALSE;
        else
            echo $content_top;
    ?>
]]></add>
```

With the previous lines it replaces all the information contained on the <*search*> tag to the lines on the <*add*> tag. In this case it will look for the variable *$content_top* and if its value contains the text *'<h2>3DViewer</h2>'* it will make the variable *$hide_pc=FALSE*. By means of this, it displays the 3D Plugin when the value of *$hide_pc* is not true. There are some further lines of code to complete the operation of modifying the basic layout of a product. Everything is done in the file install.xml in the root folder of the plugin.

Once all the alterations to the code are detailed and the modification is installed, there are a couple more of steps in order to have the Viewer in operation as a plugin for OpenCart; since, as mentioned before, the 3D Viewer is only displayed when the legend *3DViewer* is printed on top of the screen. First in the admin panel in OpenCart, in the path "Extensions/Home/Extensions", in the dropdown menu "Choose the extension type", there is the option "Modules" where is possible to install the Module "HTML Content" by clicking on install. See Figure 5.

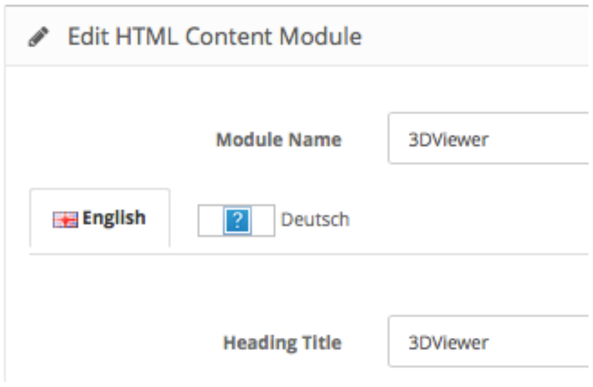


After you have installed and configured a module you can add it to a layout **here**!

Module Name	Action
Account	
Affiliate	
Banner	
Bestsellers	
Carousel	
Category	
Divido Product Page Calculator	
Featured	
Filter	
Google Hangouts	
HTML Content	

Figure 5 Install "HTML Content" Extension

Then, when clicked in "Edit", it is possible to write the text '*3D Viewer'* in the "Module Name" and "Heading Title" for every language, like in Figure 6. This text is in charge of letting the OpenCart modification know which product has enabled the 3D View.


Edit HTML Content Module

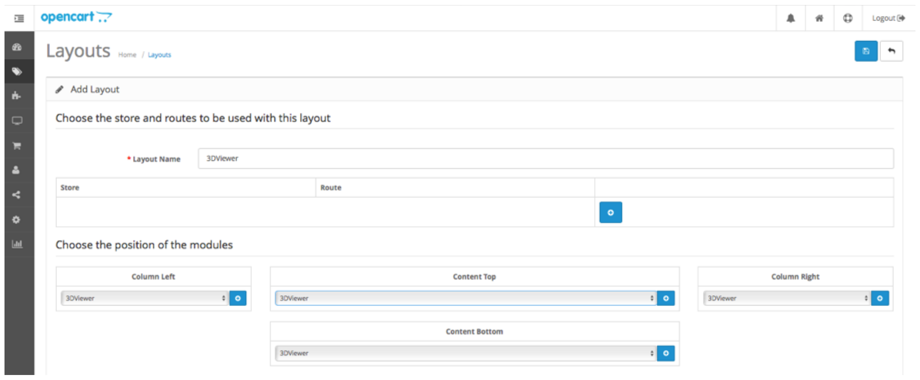
Module Name: 3DViewer

English | Deutsch

Heading Title: 3DViewer

Figure 6 Configuration of HTML Module

After the module is configured, it is possible to create a new layout that further is associated to the 3D Viewer. To do this, in the section “Layouts”, by pressing in “Add New” on the top right side of the screen, the “Layout Name” “3DViewer” is typed and configured for every content (left, top, right and bottom) in the dropdown menu the option “3DViewer”, similar to Figure 7.



Add Layout

Choose the store and routes to be used with this layout

* Layout Name: 3DViewer

Store	Route

Choose the position of the modules

Column Left: 3DViewer

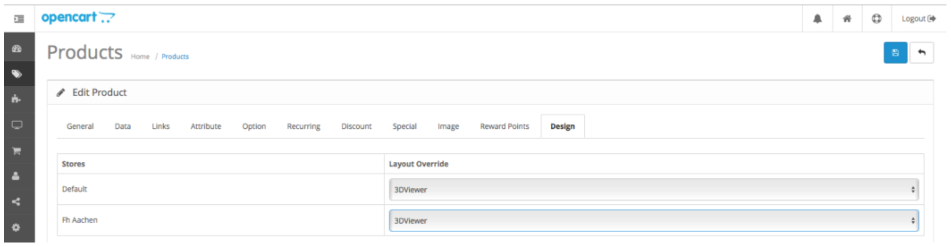
Content Top: 3DViewer

Column Right: 3DViewer

Content Bottom: 3DViewer

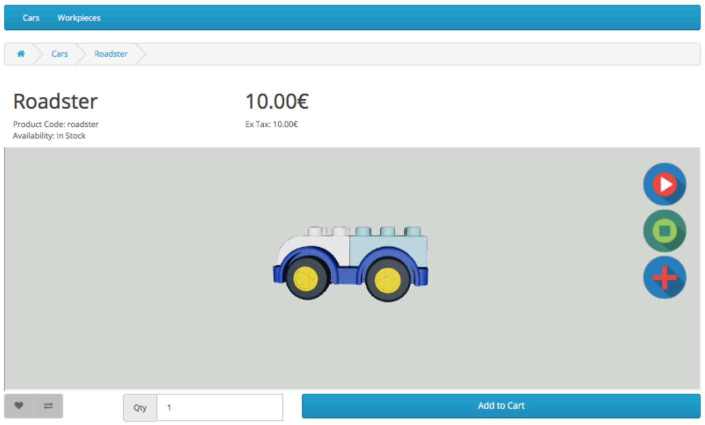
Figure 7 Configuration of the Layout.

Straightaway, the plugin is correctly configured and the layout is prepared to be related to any product. The next step is to override this layout to any Lego Duplo car to enable the 3D Visualization. In the “Product Configuration”, on the tab “Design”, there is the option “Layout Override”, where by selecting the option “3DViewer” the 3D visualization is activated as shown in Figure 8. This layout ensures that now the customization of a product looks similar to Figure 9.



Stores	Layout Override
Default	3DViewer
Fh Aachen	3DViewer

Figure 8 Enabling the Plugin to a certain Product



Roadster

10.00€

Product Code: roadster
Availability: In Stock

Ex Tax: 10.00€

1

Figure 9 Implementation of OpenCart with the Plugin enabled

Now, the user is able to rotate, zoom in, zoom out, add or delete components as well as change the color of any piece by merely clicking on it and selecting from one of the available colors, getting an immediate perspective of the Lego Duplo car.

Subsequently, when the user makes all the configurations and is satisfied with the product, it is only necessary to press the "Add to Cart" button and the system will process the order and add it to the “Shopping Cart”.

System 2 - Translator

With the previous configurations, OpenCart shows any product to the Viewer where the Layout “3DViewer” is enabled; however, it is necessary to send the template definition of the Product in use to the Viewer. This system is the one in charge of translating from “System 4 - IoT Adapter” where the administrator properly defines each template in the database under the table *opencart_actions*, to “System 3 - Viewer”, where it generates the input JSON that is described in Section System 3 - Viewer.

The “IoT Adapter” provides a REST Server with some queries for specific purposes. In this case, by making a request to “/api/oc_actions/<product_id>”, it is possible to get a JSON container with all the parameters for the template of a car. The translator reads this JSON container and transforms it into another JSON container that is more standard for the viewer, bringing only the necessary information to display something built with Lego Duplo bricks. The actions to achieve this translation and to provide interaction among the Viewer, the IoT Adapter, and OpenCart are described in the Sequence Diagram on Figure 10 Sequence Diagram System 2 - Translator. It is important to clarify that all the functions mentioned in this diagram are contained in the Script “translator.js”, previously mentioned in Figure 4 Content of the folder “3DViewer.ocmod”. The last point to clarify is that it is expected, that both the OpenCart installation and the REST Server are on the same server, consequently sharing the same IP Address. When this is not the case, it is essential to change the variable *api* from its default value *‘/api/oc_actions/’* to a value that includes the IP Address and does not refer only to the root directory of the server.

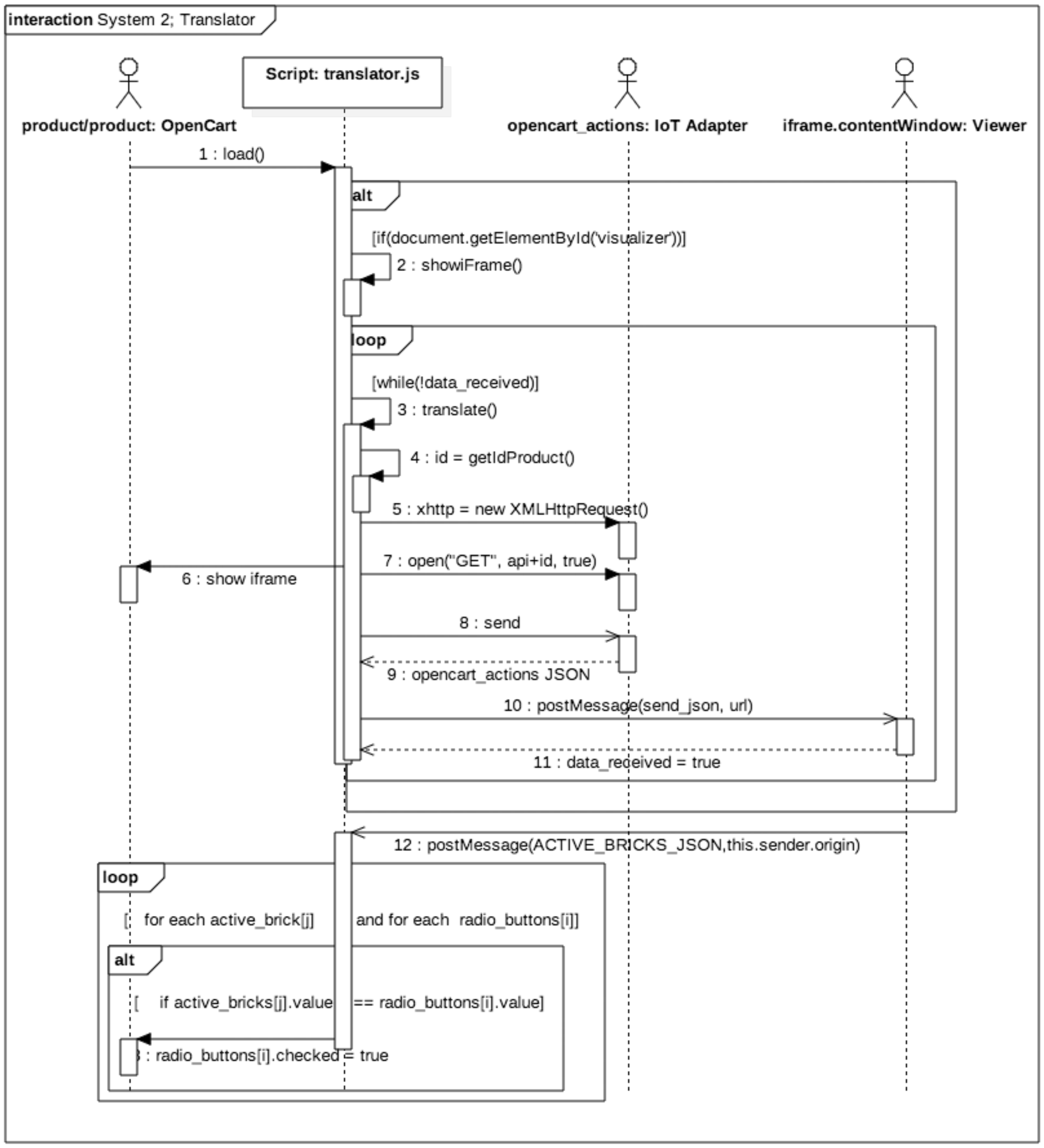
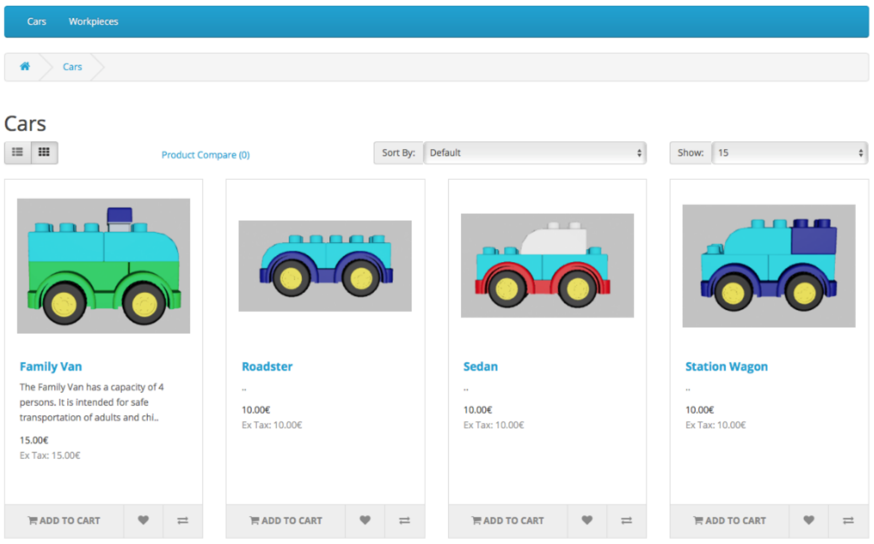

interaction System 2; Translator
Script: translator.js
product/product: OpenCart
opencart_actions: IoT Adapter
iframe.contentWindow: Viewer
1 : load()
alt
[if(document.getElementById('visualizer'))]
2 : showiFrame()
loop
[while(!data_received)]
3 : translate()
4 : id = getIdProduct()
5 : xhttp = new XMLHttpRequest()
6 : show iframe
7 : open("GET", api+id, true)
8 : send
9 : opencart_actions JSON
10 : postMessage(send_json, url)
11 : data_received = true
12 : postMessage(ACTIVE_BRICKS_JSON,this.sender.origin)
loop
[for each active_brick[j] and for each radio_buttons[i]]
alt
[if active_bricks[j].value == radio_buttons[i].value]
: radio_buttons[i].checked = true

Figure 10 Sequence Diagram System 2 - Translator

The previous sequence diagram on Figure 10 presents the normal flow of information in this translator, nevertheless it is crucial to clear some things out to make it more comprehensible. First, the user from OpenCart, selects a product on the screen that looks like Figure 11. This opens the path "/product/product", and in consequence, the "Even Listener" that is initialized in the "translator.js" triggers (sequence 1), showing the *iframe* (sequence 2), corroborating whether the visualizer was enabled or not.

The function *translate()* (sequence 3) is the one in charge of making the request to the REST Server (Sequences 4-8). In Sequence 9, receives an answer with a JSON container including the template definition, similar to the one in Figure 12. This JSON container is an example of the response for the product "roadster" with the *id = 3*.

By executing some algorithms, it transforms the information into another JSON container that is more understandable for the Viewer. An example of this JSON container is shown in Figure 22.



Cars

Family Van
The Family Van has a capacity of 4 persons. It is intended for safe transportation of adults and chi..
15.00€
Ex Tax: 15.00€

Roadster
..
10.00€
Ex Tax: 10.00€

Sedan
..
10.00€
Ex Tax: 10.00€

Station Wagon
..
10.00€
Ex Tax: 10.00€

Figure 11 Products List

```
[{"oc_product":3,"oc_option":13,"state":10,"payload":["6048908"
  ,0,0,-1,0],"item":{"code":"6048908","name":"Blue","keywords"
  :{"webgl_brickname":"chassis","webgl_color":"blue"}},"amount"
  :1},{"oc_product":3,"oc_option":15,"state":11,"payload"
  :["6048907",0,0,-1,0],"item":{"code":"6048907","name":"Red"
  ,"keywords":{"webgl_brickname":"chassis","webgl_color"
  :"red"}},"amount":1},{"oc_product":3,"oc_option":46,"state"
  :40,"payload":["6138111",-3,0,0,2],"item":{"code":"6138111"
  ,"name":"White","keywords":{"webgl_brickname":"2x3x2-bow"
  ,"webgl_color":"white"}},"amount":1},{"oc_product":3
  ,"oc_option":34,"state":41,"payload":["6138111",-3,0,0,2]
  ,"item":{"code":"6138111","name":"Lightblue","keywords"
  :{"webgl_brickname":"2x3x2-bow","webgl_color":"lightblue"}}
  ,"amount":1},{"oc_product":3,"oc_option":54,"state":50
  ,"payload":["6058256",-1,0,2,0],"item":{"code":"6058256"
  ,"name":"blue EmergancyLight","keywords":{"webgl_brickname"
  :"emergencylight","webgl_color":"blue","a":"B","b":"C"}}
  ,"amount":1},{"oc_product":3,"oc_option":56,"state":51
  ,"payload":["6058259",-1,0,2,0],"item":{"code":"6058259"
  ,"name":"orange Emergancy Light","keywords"
  :{"webgl_brickname":"emergencylight","webgl_color":"orange"}}
  ,"amount":1},{"oc_product":3,"oc_option":null,"state":20
  ,"payload":["6138409",3,0,0,0],"item":{"code":"6138409"
  ,"name":"Lightblue","keywords":{"webgl_brickname":"2x3x2"
  ,"webgl_color":"lightblue"}},"amount":1}]
```

Figure 12 Response by executing in the REST Server /api/oc_actions/3

Once the new JSON is generated, the script sends an asynchronous message to the Viewer (sequence 10) using the function:

iframe.contentWindow.postMessage(send_json, url);

Because the type of communication for this case is asynchronous, it is not possible to get a direct handshake of the systems; therefore, the way to bear out that the message was received by the Viewer, is by sending another asynchronous message (sequence 11) with the value *true* that eventually is received by the script and stored under the variable *data_received*.

To solve this problem of working asynchronously while the loading screen is displayed, just as depicted in Figure 13, the script is sending a message containing the JSON (sequence 10) every 5 seconds until the variable *data_received* changes its value to *true* or it reaches the timeout that is 30 seconds. After that period, most likely there is an error with the communication, and in consequence, the system displays an error message like in Figure 14. Another reason for this to happen is when there is not a definition of the template in the database, or as mentioned before, the REST API Server IP is in a different server as OpenCart.

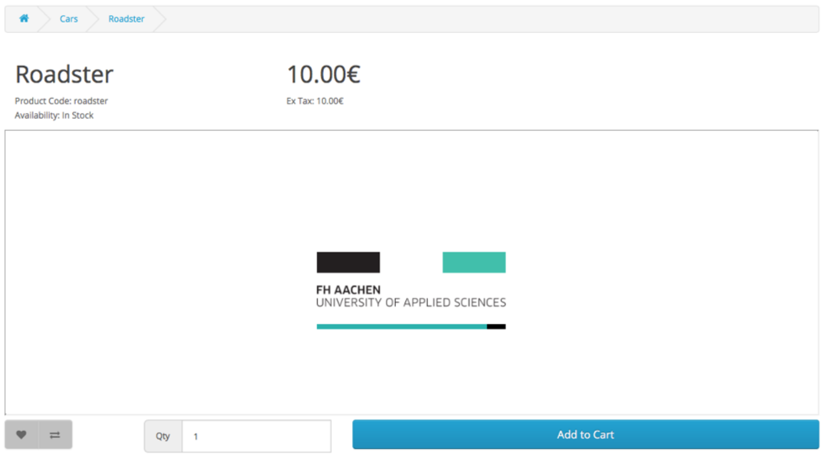


Figure 13 Loading Screen

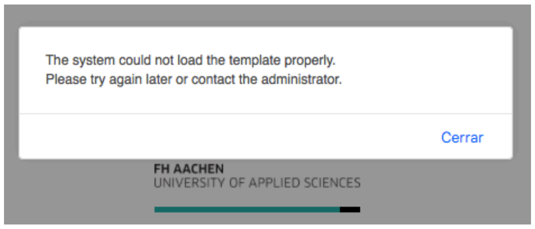

The system could not load the template properly.
Please try again later or contact the administrator.

Figure 14 Error: Timeout

If the message is never shown and the Loading Screen gets to 100% and then is disabled, it means that the communication among all systems was established correctly and is possible to start with the customization. Each time the user makes a modification to the car, e.g. he modifies the color of a piece, it deletes a brick or any other alteration, the Viewer sends a JSON container with the values of the radio buttons that should be activated to the translator, similar to:

{active_bricks: [{value: 14}, {value: 43}, {value: 12}]}

The previous JSON container, includes some random values to exemplify. With this information, the script looks into the values of every radio button and if it matches the criteria it turns the *checked* property to *true*. This process is similar as if the user would manually click on the radio buttons and makes possible to update the changes when the “Add to Cart” button is pressed.

System 3 - Viewer

This system is the one in charge of displaying a Lego Duplo car, or actually anything that is built using Lego Duplo bricks. The entire Viewer was programmed using the 3D game-engine PlayCanvas; hence, the programming language is JavaScript (JS). Since JS is an object-oriented language, it is always crucial to illustrate the behavior of the classes to have a better perception of how a system works. PlayCanvas provides some extra classes that makes it more manageable for the developer to use a 3D environment; however, to understand better the purpose, distribution, and hierarchy of the classes it is relevant to explain some basic concepts about this development platform.

The first concept is how the objects are created. In Figure 15 on the left side of the screen, there are the so-called “Entities”. These entities refer to the physical objects with whom is possible to interact. An example of this is the lights, a Lego Duplo brick or a button. These objects have a set of attributes that can be modified via code, and they are organized using a hierarchy setup, all of them having a unique parent, but in some cases one or several children. Consequently, each physical object is represented as an instance of a Class with the same name, in Figure 17 there is the Class Diagram for all the entities of the system. This diagram does not take into consideration the rest of the Classes that the system has.

To modify the properties of the entities, to change their relationship, to duplicate or delete them or to execute any other function that involves altering its attributes, each entity can contain any amount of scripts attached to it. This way of assigning scripts is exhibited in Figure 15 on the right side of the screen, where some of the associated scripts for the entity *Root* are shown. Every script is interpreted as a class, and it is instanced only one time when the program is started or when the entity is cloned.

As it can be observed in Figure 17, most of the classes contain attributes related to other classes, e.g., The class *Root* has the attribute *configureCar* with the data type *ConfigureCar*. These kinds of variables have the purpose of simplifying the usage of the objects, keeping the unique instance of each class stored in them.

Therefore, every script is related to at least one entity, and each of them can interact with any other. Figure 18, shows a Class Diagram that represents the relations among the scripts and entities of the system.

To have a better comprehension of the purpose of each script, a particular type of class documentation was created, similar to the Oracle's Javadoc[1]. Every function and variable in the code has the same structure for the comments, like the ones in Figure 16. This arrangement facilitates the comprehension of the purpose of each method. This style of comments also helps in the way the documentation is created, in Annexure 3 there is an example of the report for the Class *ConfigureCar* inside the package *CarConfiguration*. The packages are just a reference or a way to group some classes that have similar purposes, e.g., providing event listeners to the screen. This documentation is available for the future developers in an HTML format, so when they give maintenance or an upgrade to the code, it is entirely clear and accessible to modify.

[1] *http://www.oracle.com/technetwork/articles/java/index-137868.html*

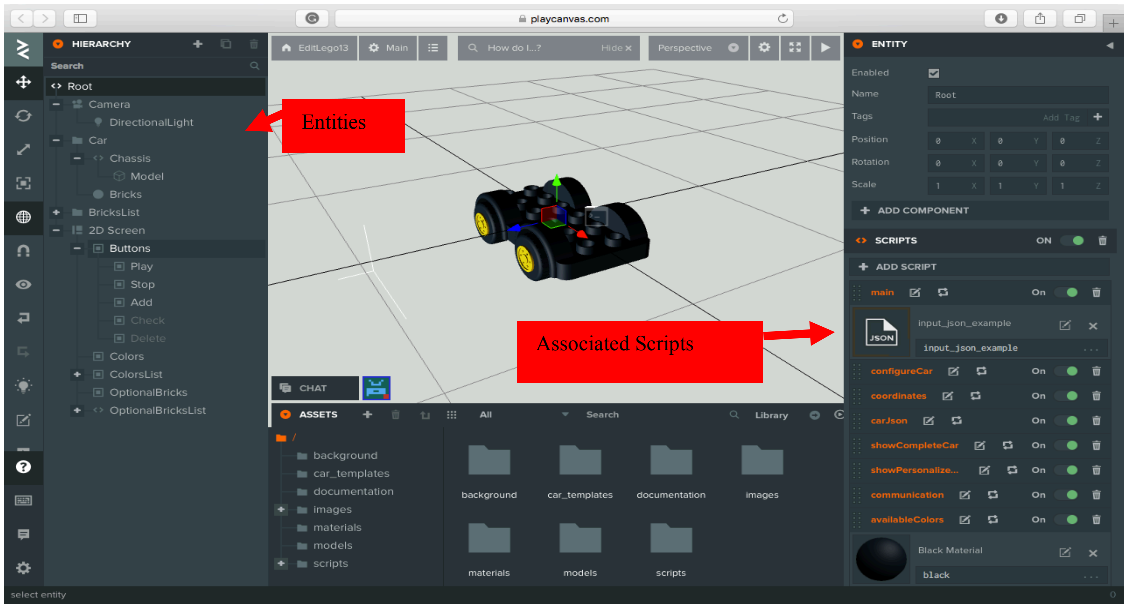
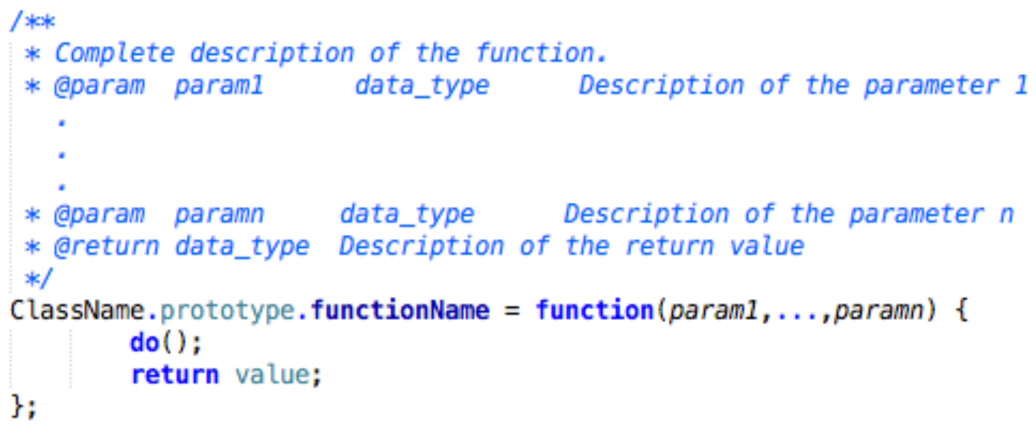


Figure 15 PlayCanvas Development Framework



```
/**
 * Complete description of the function.
 * @param  param1      data_type      Description of the parameter 1
   .
   .
   .
 * @param  paramn     data_type      Description of the parameter n
 * @return data_type  Description of the return value
 */
ClassName.prototype.functionName = function(param1,...,paramn) {
        do();
        return value;
};
```

Figure 16 Comments on a function

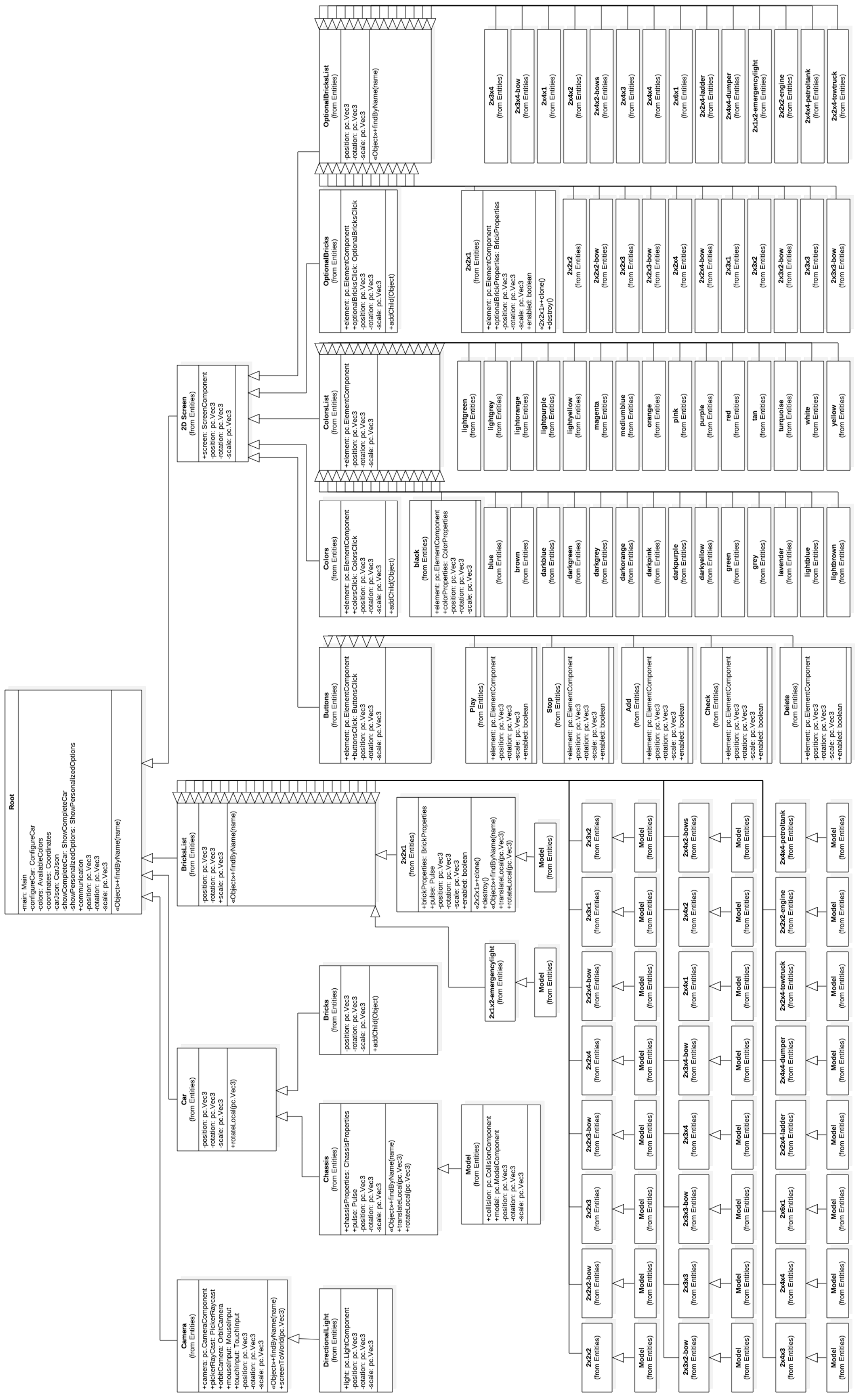


Figure 17 Class Diagram: Entities

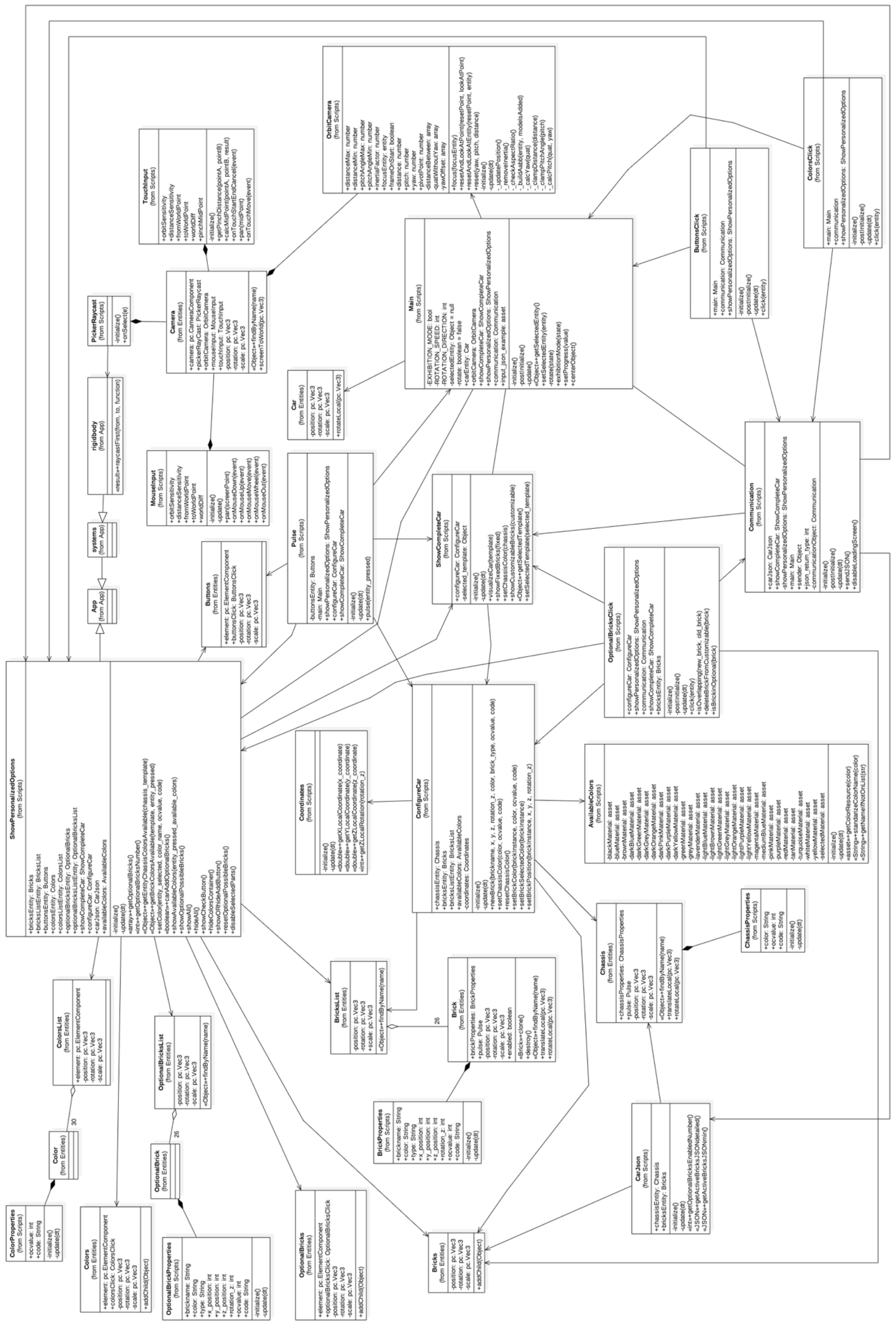


Figure 18 Class Diagram: Entities-Scripts Relations

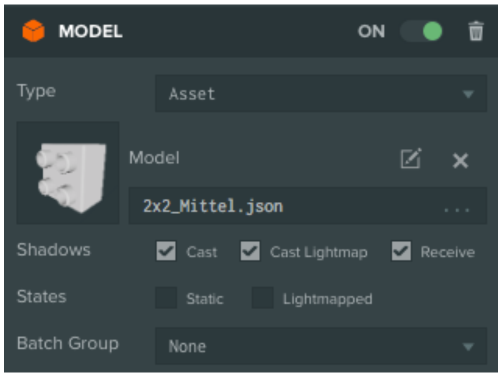
Something important to clarify in here is how an entity is related to a 3D model. To start with, every Lego Duplo part is represented as an entity and consequently as an object (e.g., 2x2x2, 2x4x4-petroltank) and every one of them contains a child named Model. This child holds an element of the type *pc.ModelComponent* like the one in Figure 19.

To upload a 3D model as an asset, first, a CAD file normally with the ".obj" extension has to be converted to an ".fbx" file using the Autodesk FBX® Converter software previously described. Then it can be uploaded to PlayCanvas and is going to be automatically converted to a JSON file. Finally, this JSON file is the one that can be associated with the *pc.ModelComponent* of each piece.

The behavior of all the entities that are children of the Optional Bricks List and Colors List is similar, but instead of having a *pc.ModelComponent* related to their child, they have a *pc.ElementComponent* associated directly with the entity and this component contains a static image.

One issue that was faced when developing the software is that the asset model and the asset image were not possible to change them via code. Therefore, instead of having a general entity for a brick or a color, the use of an entity per type of piece or per variety of color was necessary.

In the Class Diagram in Figure 18 the child of BricksList is an object named Brick, for OptionalBricksList is OptionalBrick and for ColorsList is Color, this was done to simplify the diagram but this manage of entities, in fact, is a representation of all the possible bricks or colors. These generalizations of the specific bricks and colors objects are represented as an aggregation relationship in the Class Diagram in Figure 18.


MODEL ON

Type: Asset

Model: 2x2_Mittel.json

Shadows: ✓ Cast ✓ Cast Lightmap ✓ Receive

States: Static Lightmapped

Batch Group: None

Figure 19 pc.ModelComponent for the child Model of the entity 2x2x2

As it can be seen in the previous Class Diagrams in Figure 17 and Figure 18, the system has around 20 scripts and 105 entities, all associated with other classes. Despite the fact that in the Class Diagram in Figure 18 the relations are shown, this diagram exhibits only the connections statically, making difficult to follow the path of the shared information or to know which methods were used.

To simplify the previous dilemma, the system is split into two subsystems that go along with the natural process of interaction with the program.

Subsystem 3.1

Figure 2 shows three cases for the Viewer. The Subsystem 3.1 englobes the case “Receives JSON Template” and “Show Lego Duplo Car”. The reason to combine them is that each time the Viewer is opened; these both actions are executed only once. Figure 20 presents a vague representation of the trail of operations.

Every time the Viewer is opened, it waits until a new message is received. When this happens and if the message is correct, it disables the loading screen, shows the car, habilitates the buttons and finally sends back a confirmation to the window that opened it.

The flow of activities for this particular interaction is displayed in Figure 20. However, it does not explain in detail how the system’s code is structured. For this purpose, the Sequence Diagrams are useful, because they include the exact classes, entities, functions, and parameters used in the code. Figure 21 corresponds to the same functionality as the Activity Diagram in Figure 20, but in a Sequence Diagram.

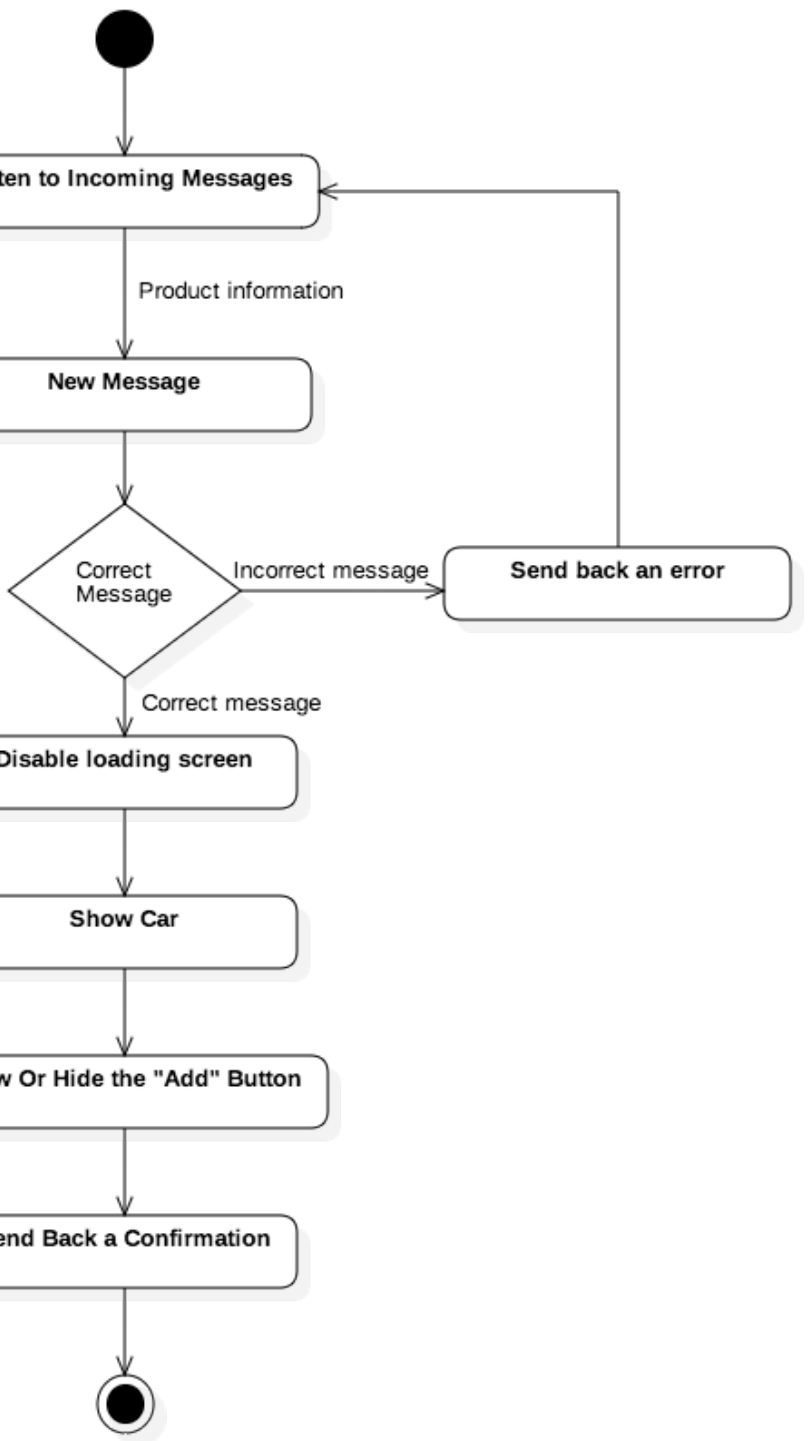


Figure 20 Activity Diagram: Subsystem 3.1

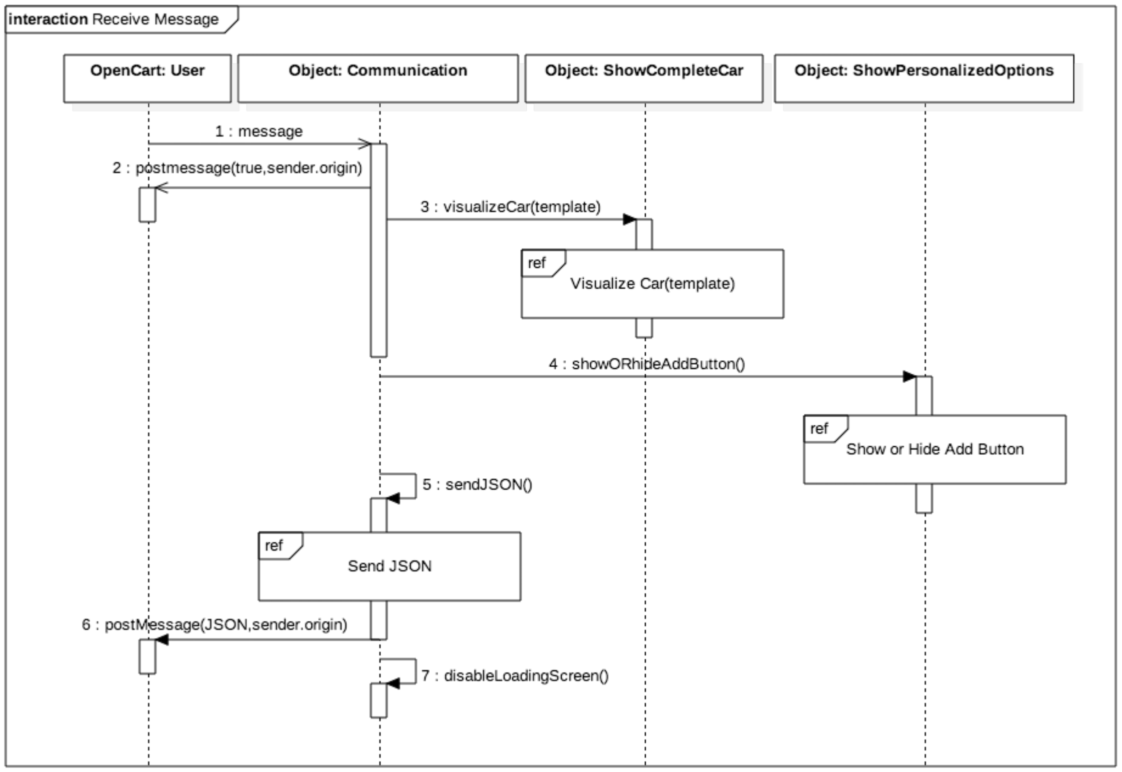

interaction Receive Message
OpenCart: User
Object: Communication
Object: ShowCompleteCar
Object: ShowPersonalizedOptions
1 : message
2 : postmessage(true,sender.origin)
3 : visualizeCar(template)
ref
Visualize Car(template)
4 : showORhideAddButton()
ref
Show or Hide Add Button
5 : sendJSON()
ref
Send JSON
6 : postMessage(JSON,sender.origin)
7 : disableLoadingScreen()

Figure 21 Sequence Diagram: Subsystem 3.1

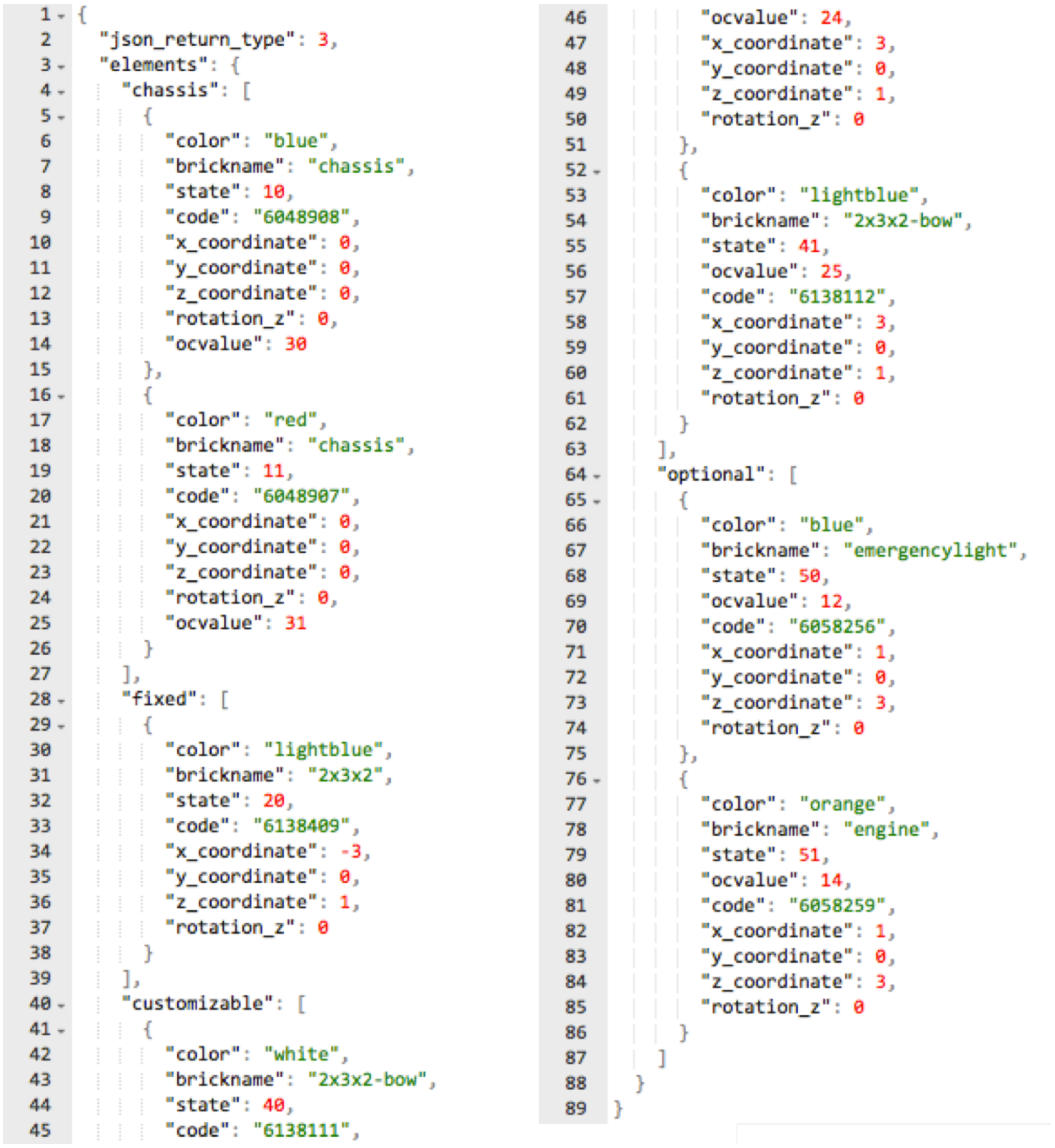
Sequence diagrams can be too complex in the design, so that instead of making the system more comprehensive, they just confuse more. Besides, some of the sub-sequences are repeated many times, and that is the reason why in UML 2.0 the reference to sub-sequence diagrams is possible. Figure 21 reduces some procedures into three sub-diagrams instead of a complex and big one. The functionalities of each sub-sequence is described in this section.

The first two actions in the Activity Diagram in Figure 20 are to listen to incoming messages until a new message is received.

To activate this listener, the following function is called inside the method *Communication.prototype.postInitialize* on the Script Communication.js:

window.addEventListener("message", function (event) {}

Inside this message, the enclosed data is only a JSON container, similar to the one in Figure 22. To send this message, the type of communication is via the JavaScript function *window.postMessage()*. The JSON container shown in the example corresponds to a Lego Duplo car of the type “Roadster”.



```
1  {
2    "json_return_type": 3,
3    "elements": {
4      "chassis": [
5        {
6          "color": "blue",
7          "brickname": "chassis",
8          "state": 10,
9          "code": "6048908",
10         "x_coordinate": 0,
11         "y_coordinate": 0,
12         "z_coordinate": 0,
13         "rotation_z": 0,
14         "ocvalue": 30
15       },
16       {
17         "color": "red",
18         "brickname": "chassis",
19         "state": 11,
20         "code": "6048907",
21         "x_coordinate": 0,
22         "y_coordinate": 0,
23         "z_coordinate": 0,
24         "rotation_z": 0,
25         "ocvalue": 31
26       }
27     ],
28     "fixed": [
29       {
30         "color": "lightblue",
31         "brickname": "2x3x2",
32         "state": 20,
33         "code": "6138409",
34         "x_coordinate": -3,
35         "y_coordinate": 0,
36         "z_coordinate": 1,
37         "rotation_z": 0
38       }
39     ],
40     "customizable": [
41       {
42         "color": "white",
43         "brickname": "2x3x2-bow",
44         "state": 40,
45         "code": "6138111",
46         "ocvalue": 24,
47         "x_coordinate": 3,
48         "y_coordinate": 0,
49         "z_coordinate": 1,
50         "rotation_z": 0
51       },
52       {
53         "color": "lightblue",
54         "brickname": "2x3x2-bow",
55         "state": 41,
56         "ocvalue": 25,
57         "code": "6138112",
58         "x_coordinate": 3,
59         "y_coordinate": 0,
60         "z_coordinate": 1,
61         "rotation_z": 0
62       }
63     ],
64     "optional": [
65       {
66         "color": "blue",
67         "brickname": "emergencylight",
68         "state": 50,
69         "ocvalue": 12,
70         "code": "6058256",
71         "x_coordinate": 1,
72         "y_coordinate": 0,
73         "z_coordinate": 3,
74         "rotation_z": 0
75       },
76       {
77         "color": "orange",
78         "brickname": "engine",
79         "state": 51,
80         "ocvalue": 14,
81         "code": "6058259",
82         "x_coordinate": 1,
83         "y_coordinate": 0,
84         "z_coordinate": 3,
85         "rotation_z": 0
86       }
87     ]
88   }
89 }
```

Figure 22 Configuration JSON container for the "Roadster" model

The basic structure of the configuration JSON container is an object with two elements. The first component is the variable *json_return_type* (line 2 in Figure 22). This variable informs the Viewer what kind of information it requires to receive as a response each time a change is made on the Lego Duplo car. The data type is an integer with the possible values:

0. Do not send anything back (Demo mode).

1. Minified JSON container: This only returns the values of the radio buttons that are activated in the configuration (ocvalue), so that they can be activated in the HTML page; consequently, the configuration can be added to the shopping cart.
2. Detailed configuration JSON container: Returns all the parameters of the configured bricks.
3. Prints in the console the Minified JSON container.
4. Prints in the console the Detailed configuration JSON container.

The second element is an array called elements (line 3 in Figure 1). Here all the pieces are separated into four items depending on the possibility to edit, delete or configure them. It is essential that the JSON always contains the four elements, even if they are empty, to prevent the system from crashing. These items are:

- **chassis** (line 4 in Figure 22): This element contains an object array with the definition of all the possible options for the chassis. Each different chassis should be described as a single element with all the properties that are going to be detailed later. If this array is empty, the Viewer won't show a chassis, but the rest of the bricks will be displayed.
- **fixed** (line 28 in Figure 22): This element contains an object array with the definition of all the bricks that are “fixed”. This means that the color cannot be changed and the item cannot be deleted from the Lego Duplo car. When the Viewer is started, all the bricks defined here will be shown in the car.
- **customizable** (line 40 in Figure 22): This element contains an object array with all the bricks that are customizable. That means, that it is possible to change their appearance, but they cannot be deleted from the car. When the Viewer is started, all the bricks defined here with a state with a value multiple of 10 will be shown in the car (e.g., 20, 30, 40). This is used to prevent the same brick to be displayed twice with different colors.
- **optional** (line 64 in Figure 22): This element contains an object array with all the bricks that are optional. That means that their appearance can be changed and they also can be deleted from the car. When the Viewer is started and if this array is not empty, the “Add” button will be shown on the right side of the screen, and when it is pressed, it let the user to add any of the bricks defined in this container.

Once each element is created, the next step is to fill them with the information of the bricks. Each brick contains the same properties, like the ones described below.

- **color**: This property contains a string with the name of the color to be displayed. The name of the color has to be the same as any color described in Table 1 Available Colors. If the name of the color is written incorrectly or it does not exist, the system will display that brick with the default color black.
- **brickname:** Here it is referred to the type of brick. In Table 2 Available Bricks there is a full list of the pieces that the system can display. The name must be the same as the one in the column “Brick Name for Template Definition”.
- **state:** The state property is a number that associates the different color options for the same brick. The description of this property is given more in detail in section 0.
- **ocvalue:** This number is the value that normally OpenCart uses to reference a radio button for the customization of a product. If the Viewer is used in any different implementation, this value can be a random quantity, but all the pieces must have a different value.
- **code:** Unique identifier for a piece. It can be any string or a null value. The value inside here does not affect the program; it is just a reference that can be used when the detailed configuration JSON is used.
- **x_coordinate:** Translation of the brick in the x-axis. In section 0 there is a detailed description of the coordinate system for this and the following axes.
- **y_coordinate:** Translation of the brick in the y-axis.
- **z_coordinate:** Translation of the brick in the z-axis.
- **rotation_z:** Rotation of the brick in the z-axis.

As seen in the Activity Diagram in Figure 20, after the message is received, it verifies if it has the appropriate information; if so, it sends a confirmation message to the sender using the function:

communicationObject.sender.source.postMessage(true,communicationObject.sender.origin);

This way, when the sender receives the confirmation message, it stops sending the message any more. This type of acknowledgment is required since the type of communication is asynchronous.

The next step is to disable the loading screen and to visualize the car. This is done in the sub-sequence: Visualize Car (Figure 23 Sequence Diagram: Visualize Car): Hereabouts, the Lego Duplo car is displayed based on the input parameters, the program respectively executes two sub-sequences: Set the chassis color (Figure 24 Sequence Diagram: Set Chassis Color) and iteratively show and configure every Lego Duplo brick (Figure 25 Sequence Diagram: Add a Brick). The last Sequence Diagram references Figure 26 Sequence Diagram: Set Brick Color.

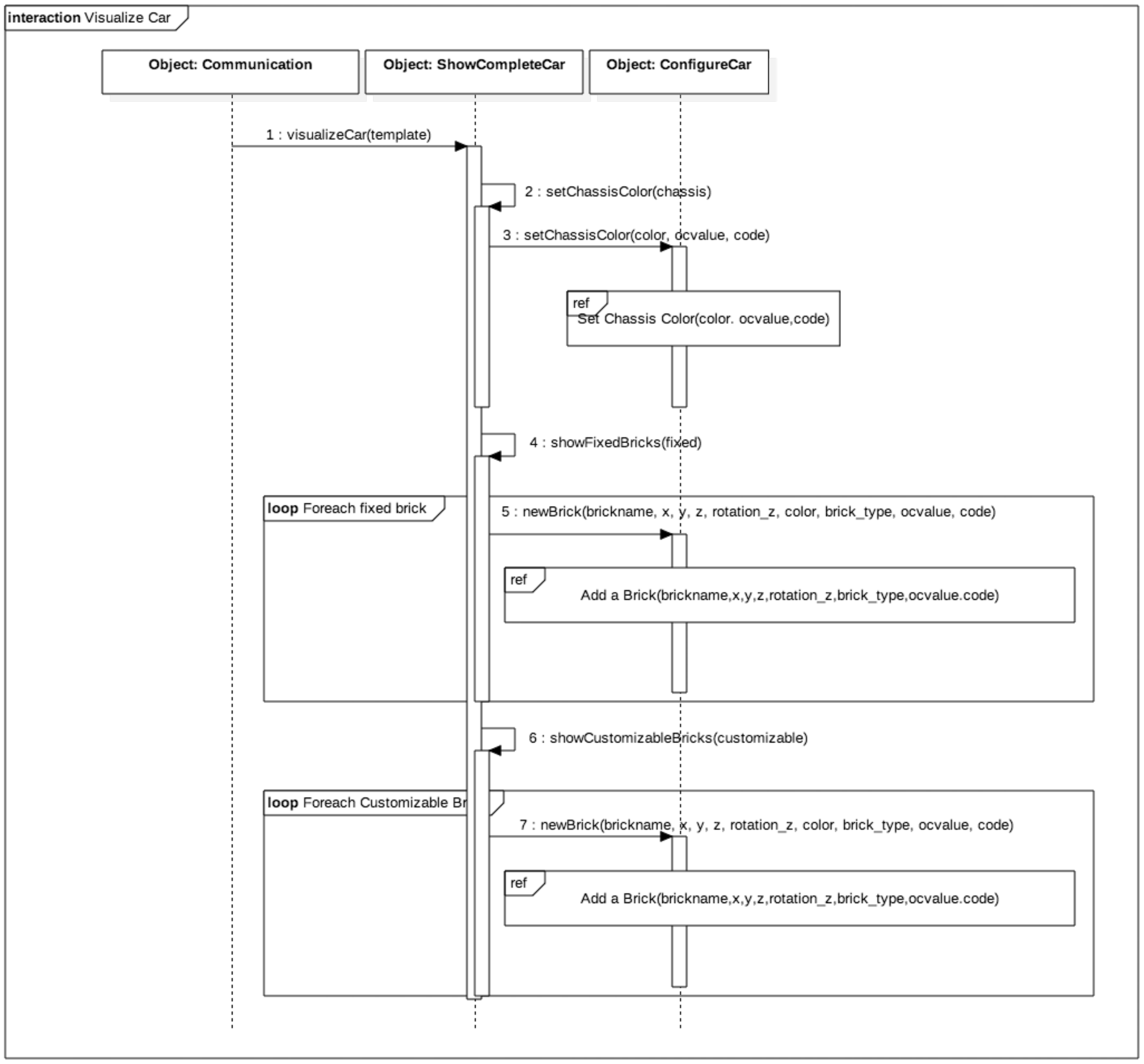

interaction Visualize Car
Object: Communication
Object: ShowCompleteCar
Object: ConfigureCar
1 : visualizeCar(template)
2 : setChassisColor(chassis)
3 : setChassisColor(color, ocvalue, code)
ref
Set Chassis Color(color. ocvalue,code)
4 : showFixedBricks(fixed)
loop Foreach fixed brick
5 : newBrick(brickname, x, y, z, rotation_z, color, brick_type, ocvalue, code)
ref
Add a Brick(brickname,x,y,z,rotation_z,brick_type,ocvalue.code)
6 : showCustomizableBricks(customizable)
loop Foreach Customizable Br
7 : newBrick(brickname, x, y, z, rotation_z, color, brick_type, ocvalue, code)
ref
Add a Brick(brickname,x,y,z,rotation_z,brick_type,ocvalue.code)

Figure 23 Sequence Diagram: Visualize Car

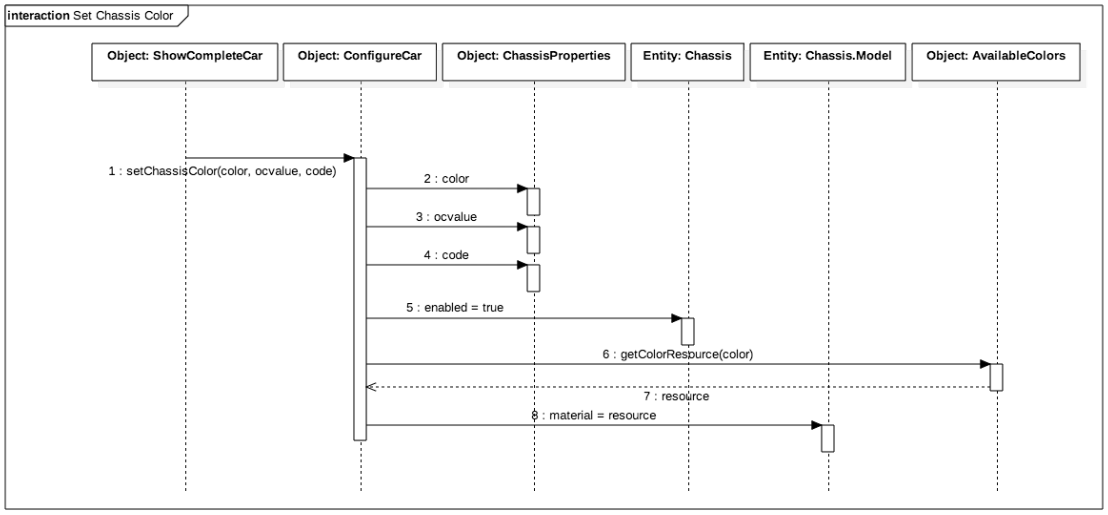


Figure 24 Sequence Diagram: Set Chassis Color

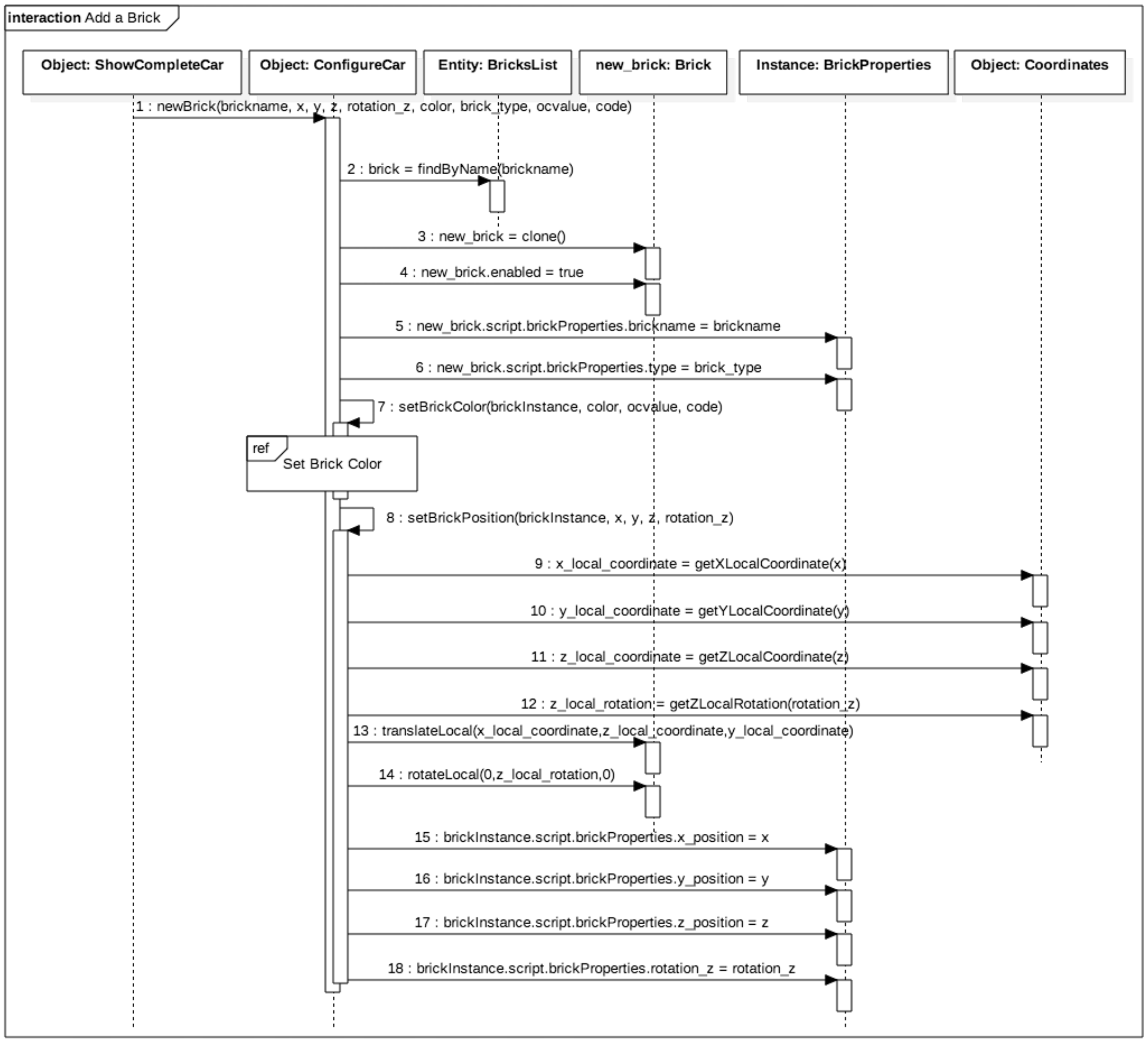

interaction Add a Brick
Object: ShowCompleteCar
Object: ConfigureCar
Entity: BricksList
new_brick: Brick
Instance: BrickProperties
Object: Coordinates
1 : newBrick(brickname, x, y, z, rotation_z, color, brick_type, ocvalue, code)
2 : brick = findByName(brickname)
3 : new_brick = clone()
4 : new_brick.enabled = true
5 : new_brick.script.brickProperties.brickname = brickname
6 : new_brick.script.brickProperties.type = brick_type
7 : setBrickColor(brickInstance, color, ocvalue, code)
ref
Set Brick Color
8 : setBrickPosition(brickInstance, x, y, z, rotation_z)
9 : x_local_coordinate = getXLocalCoordinate(x)
10 : y_local_coordinate = getYLocalCoordinate(y)
11 : z_local_coordinate = getZLocalCoordinate(z)
12 : z_local_rotation = getZLocalRotation(rotation_z)
13 : translateLocal(x_local_coordinate,z_local_coordinate,y_local_coordinate)
14 : rotateLocal(0,z_local_rotation,0)
15 : brickInstance.script.brickProperties.x_position = x
16 : brickInstance.script.brickProperties.y_position = y
17 : brickInstance.script.brickProperties.z_position = z
18 : brickInstance.script.brickProperties.rotation_z = rotation_z

Figure 25 Sequence Diagram: Add a Brick

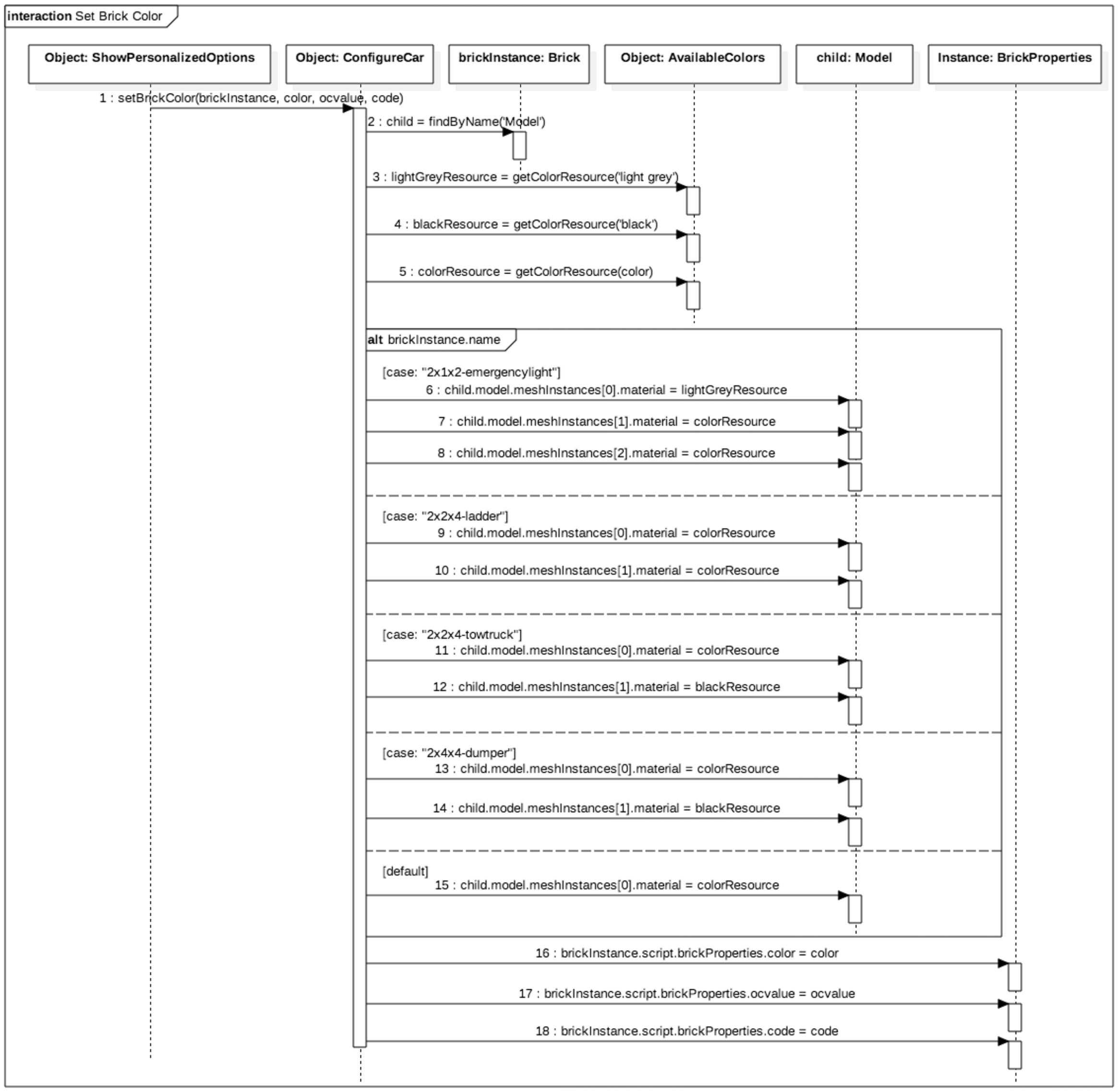

interaction Set Brick Color
Object: ShowPersonalizedOptions
Object: ConfigureCar
brickInstance: Brick
Object: AvailableColors
child: Model
Instance: BrickProperties
1 : setBrickColor(brickInstance, color, ocvalue, code)
2 : child = findByName('Model')
3 : lightGreyResource = getColorResource('light grey')
4 : blackResource = getColorResource('black')
5 : colorResource = getColorResource(color)
alt brickInstance.name
[case: "2x1x2-emergencylight"]
6 : child.model.meshInstances[0].material = lightGreyResource
7 : child.model.meshInstances[1].material = colorResource
8 : child.model.meshInstances[2].material = colorResource
[case: "2x2x4-ladder"]
9 : child.model.meshInstances[0].material = colorResource
10 : child.model.meshInstances[1].material = colorResource
[case: "2x2x4-towtruck"]
11 : child.model.meshInstances[0].material = colorResource
12 : child.model.meshInstances[1].material = blackResource
[case: "2x4x4-dumper"]
13 : child.model.meshInstances[0].material = colorResource
14 : child.model.meshInstances[1].material = blackResource
[default]
15 : child.model.meshInstances[0].material = colorResource
16 : brickInstance.script.brickProperties.color = color
17 : brickInstance.script.brickProperties.ocvalue = ocvalue
18 : brickInstance.script.brickProperties.code = code

Figure 26 Sequence Diagram: Set Brick Color

After displaying the car, the system should show or hide the "Add" button (Figure 27 Sequence Diagram: Show or Hide Add Button): This process, once it is invoked, checks if it is still possible to add optional bricks, and when it is possible, it shows a button on the screen that when pressed, displays the "Optional Bricks Picker" (where the user can select which brick to add). This method is used many times in the system, therefore the necessity of having it as a sub-sequence.

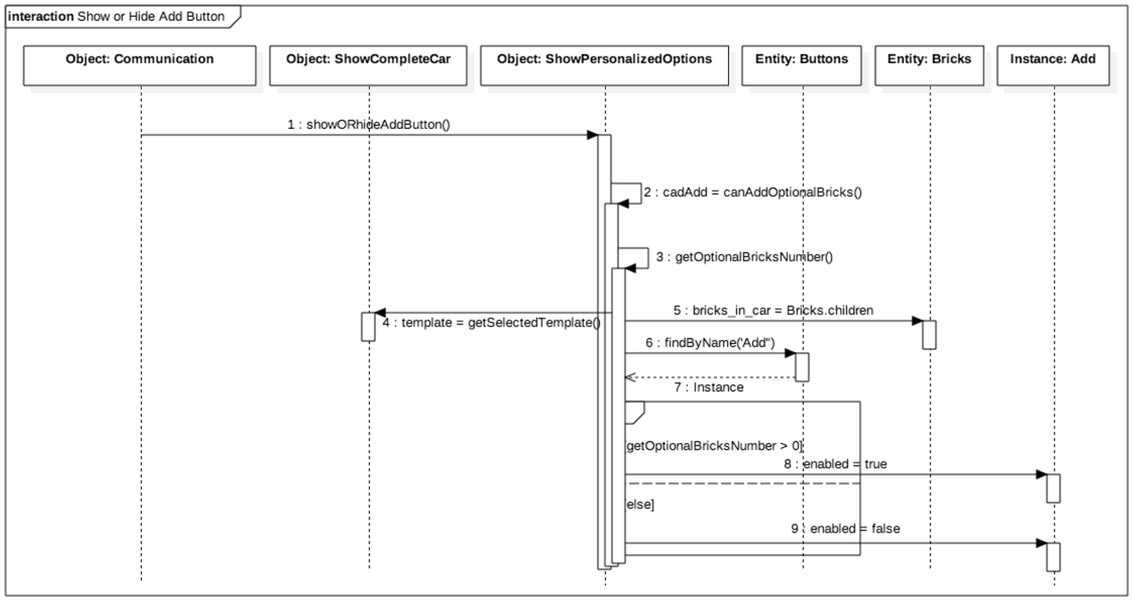

interaction Show or Hide Add Button
Object: Communication
Object: ShowCompleteCar
Object: ShowPersonalizedOptions
Entity: Buttons
Entity: Bricks
Instance: Add
1 : showORhideAddButton()
2 : cadAdd = canAddOptionalBricks()
3 : getOptionalBricksNumber()
4 : template = getSelectedTemplate()
5 : bricks_in_car = Bricks.children
6 : findByName('Add")
7 : Instance
[getOptionalBricksNumber > 0]
8 : enabled = true
[else]
9 : enabled = false

Figure 27 Sequence Diagram: Show or Hide Add Button

Finally, the system needs to send back confirmation by sending a JSON container (Figure 28 Sequence Diagram: Send JSON): When this sequence is called, depending on a particular parameter configured in the message, it gives a JSON container with the relevant data of the Lego Duplo car – such as the active bricks – back to the sender (the program that opened the viewer).

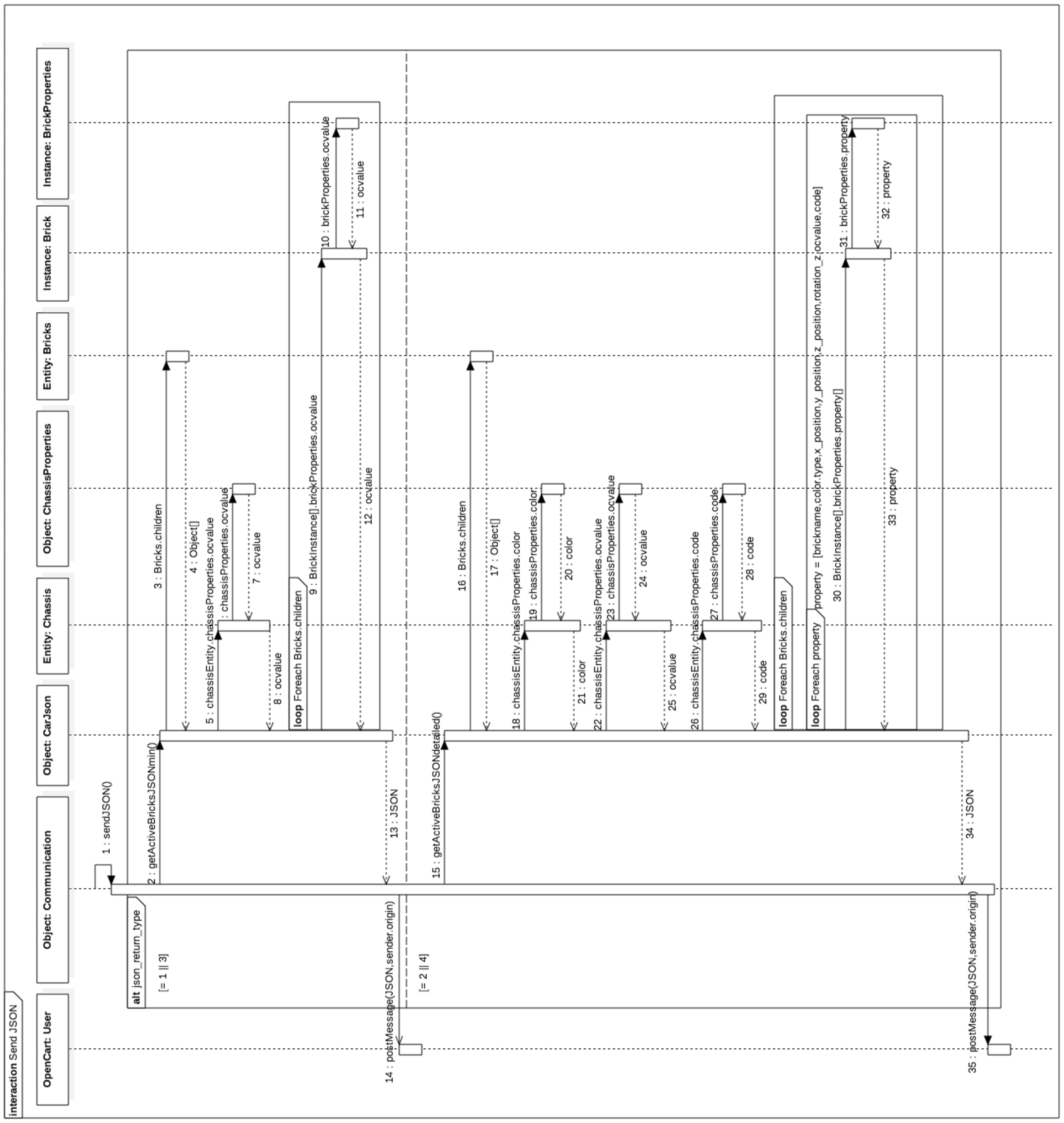

interaction Send JSON
OpenCart: User
Object: Communication
Object: CarJson
Entity: Chassis
Object: ChassisProperties
Entity: Bricks
Instance: Brick
Instance: BrickProperties
alt json_return_type
[= 1 || 3]
[= 2 || 4]
1 : sendJSON()
2 : getActiveBricksJSONmin()
3 : Bricks.children
4 : Object[]
5 : chassisEntity.chassisProperties.ocvalue
6 : chassisProperties.ocvalue
7 : ocvalue
8 : ocvalue
loop Foreach Bricks.children
9 : BrickInstance[].brickProperties.ocvalue
10 : brickProperties.ocvalue
11 : ocvalue
12 : ocvalue
13 : JSON
14 : postMessage(JSON,sender.origin)
15 : getActiveBricksJSONdetailed()
16 : Bricks.children
17 : Object[]
18 : chassisEntity.chassisProperties.color
19 : chassisProperties.color
20 : color
21 : color
22 : chassisEntity.chassisProperties.ocvalue
23 : chassisProperties.ocvalue
24 : ocvalue
25 : ocvalue
26 : chassisEntity.chassisProperties.code
27 : chassisProperties.code
28 : code
29 : code
loop Foreach Bricks.children
loop Foreach property
property = [brickname,color,type,x_position,y_position,z_position,rotation_z,ocvalue,code]
30 : BrickInstance[].brickProperties.property[]
31 : brickProperties.property
32 : property
33 : property
34 : JSON
35 : postMessage(JSON,sender.origin)

Figure 28 Sequence Diagram: Send JSON

Subsystem 3.2

This subsystem is in charge of the physical interaction with the Viewer; it detects a single touch of the user (the customer) to any part of the screen. This process can be carried out either via a mouse or a touch-screen. The methods called by both peripherals are the same. Hence, there is no distinction in which one is used. The foremost difference between this subsystem and the previous one resides in the fact that in this case there is not a unique flow of activities because the user can interact with the system in the order he pleases.

This subsystem is represented in the case "Interact with 3D Viewer" in Figure 2; notwithstanding, the case is too abstract, so it needs a broader description. Figure 29 displays a sub-case diagram for this particular case, involving all the actions the Actor" (user) can physically execute on the Viewer. Furthermore, Table 5 gives a more detailed description of each use case.

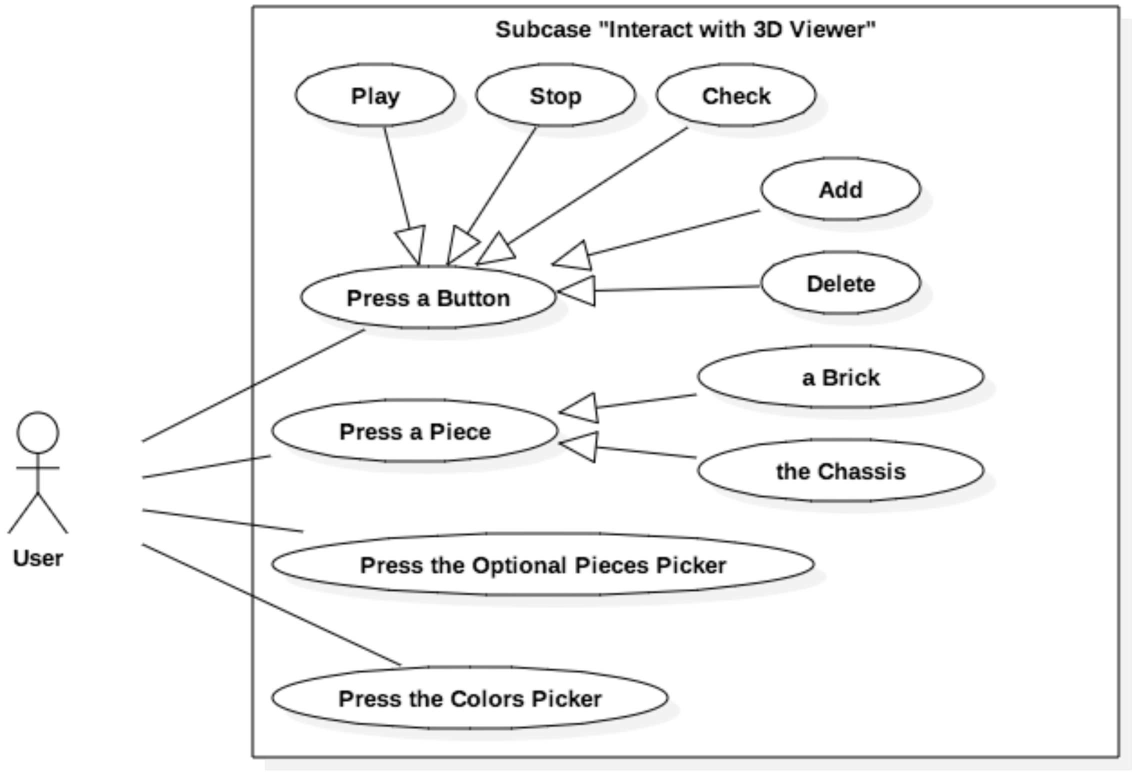


Figure 29 Use Case Diagram: User Physical Interaction

Use Case	Description
Press a Button	Any of the buttons is pressed.
Play	Centers the car on the screen, and it starts rotating it automatically.
Stop	Stops the automatic rotation.
Check	The brick that is selected (transparent) stops being selected, so the optional colors and the rest of the buttons disappears.

Add	Shows the “Optional Bricks Picker”.
Delete	Erases the selected brick from the car.
Press a Piece	The user touches any of the components of the car.
Press a Brick	The touched component is a Brick.
Press the Chassis	The touched component is the chassis.
Press Optional Bricks Picker	Adds the optional brick to the car, when the “Optional Bricks Picker” is enabled.
Press Colors Picker	Sets the color of the selected brick when the “Colors Picker” is enabled.

Table 5 Description of the Use Case Diagram: User Physical Interaction

To have a better comprehension of each use case, a description of each of them is presented.

Press a Button

Buttons provide a straightforward control for specific actions. The system has in total five buttons described next from top to bottom:

1. **Play**: Center the car on the screen, and it starts rotating the car automatically for a better visualization (button always visible). #1 in Figure 30 and Figure 31.
2. **Stop**: Stop the automatic rotation (button always visible). #2 in Figure 30 and Figure 31.
3. **Add**: It lets the user add a brick that is optional. The button is hidden when there are no bricks left to add. Otherwise when it is pressed, it displays the “Optional Bricks Picker”, where the user can add a piece to the car. #3 in Figure 30, in Figure 31 it is hidden because the only optional piece for this template was already in the car.
4. **Check**: This button will appear when any brick or the chassis is selected. When it is pressed, the brick will stop being selected, so the “Colors Picker” and the “Delete Button” disappear when it is enabled. #4 in Figure 31.
5. **Delete**: When a brick that is optional is selected, this button will appear and will let the user erase the selected brick. #5 in Figure 31.

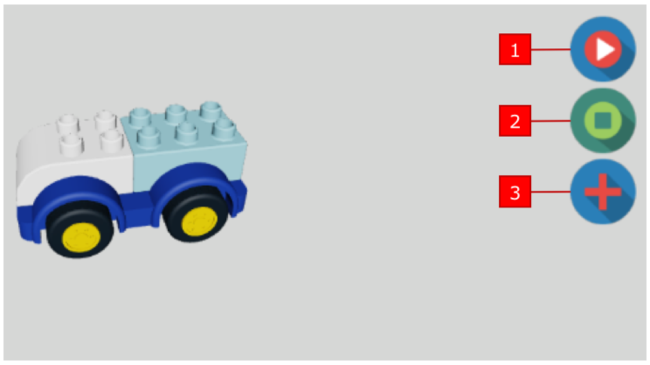


Figure 30 Viewer screen

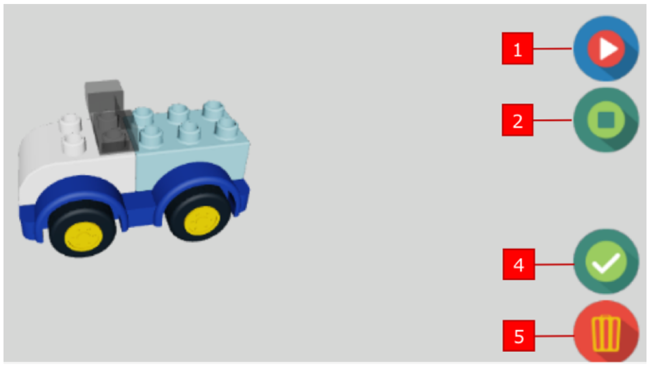


Figure 31 Viewer screen, optional piece selected

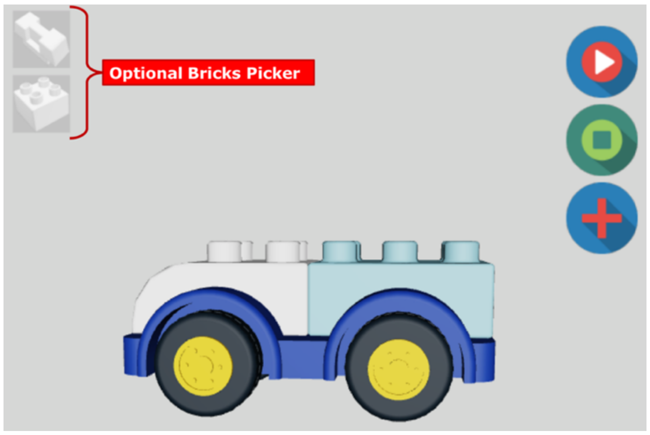
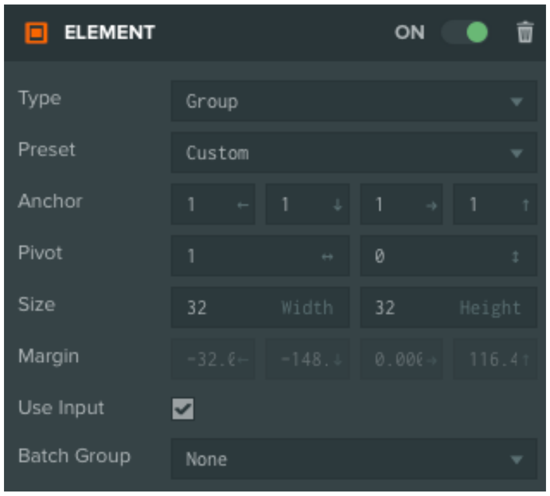


Figure 32 Optional Bricks Picker on Screen

To execute any of the previous processes, thanks to the hierarchy management in PlayCanvas, instead of providing event listeners and specific scripts for each button, this action is carried out by the parent. The entity *Buttons* has assigned an element component, like the one in Figure 33 with the option *Use Input* enabled plus a script containing the event click listener. Both elements in coordination perform the sequence that is described in Figure 34. The previous sequence diagram uses the sub-sequence diagrams “Hide All” (Figure 35), “Send JSON” (Figure 28), "Show or Hide Add Button" (Figure 27), and the sub-sequence diagram "Hide All" uses the sub-sequence diagram "Disable Selected Parts" (Figure 38) consequently.

The sequence "Hide All" basically hides all the buttons from the screen that cannot be used any more, for example the one for deleting pieces or to check. Moreover, it hides either the "Color Picker" or the "Optional Pieces Picker" and lastly, it returns any piece that was left as selected (that looked transparent) to the original color. This last sequence is carried out by the sub-sequence "Disable Selected Parts". The sequences "Send JSON" and "Show or Hide Add Button" were already described on section 0.


ELEMENT ON

Type	Group			
Preset	Custom			
Anchor	1 ←	1 ↓	1 →	1 ↑
Pivot	1 ↔		0 ↕	
Size	32 Width		32 Height	
Margin	-32.0 ←	-148. ↓	0.000 →	116.4 ↑
Use Input	✓			
Batch Group	None			

Figure 33 Element Component on Entity "Bricks"

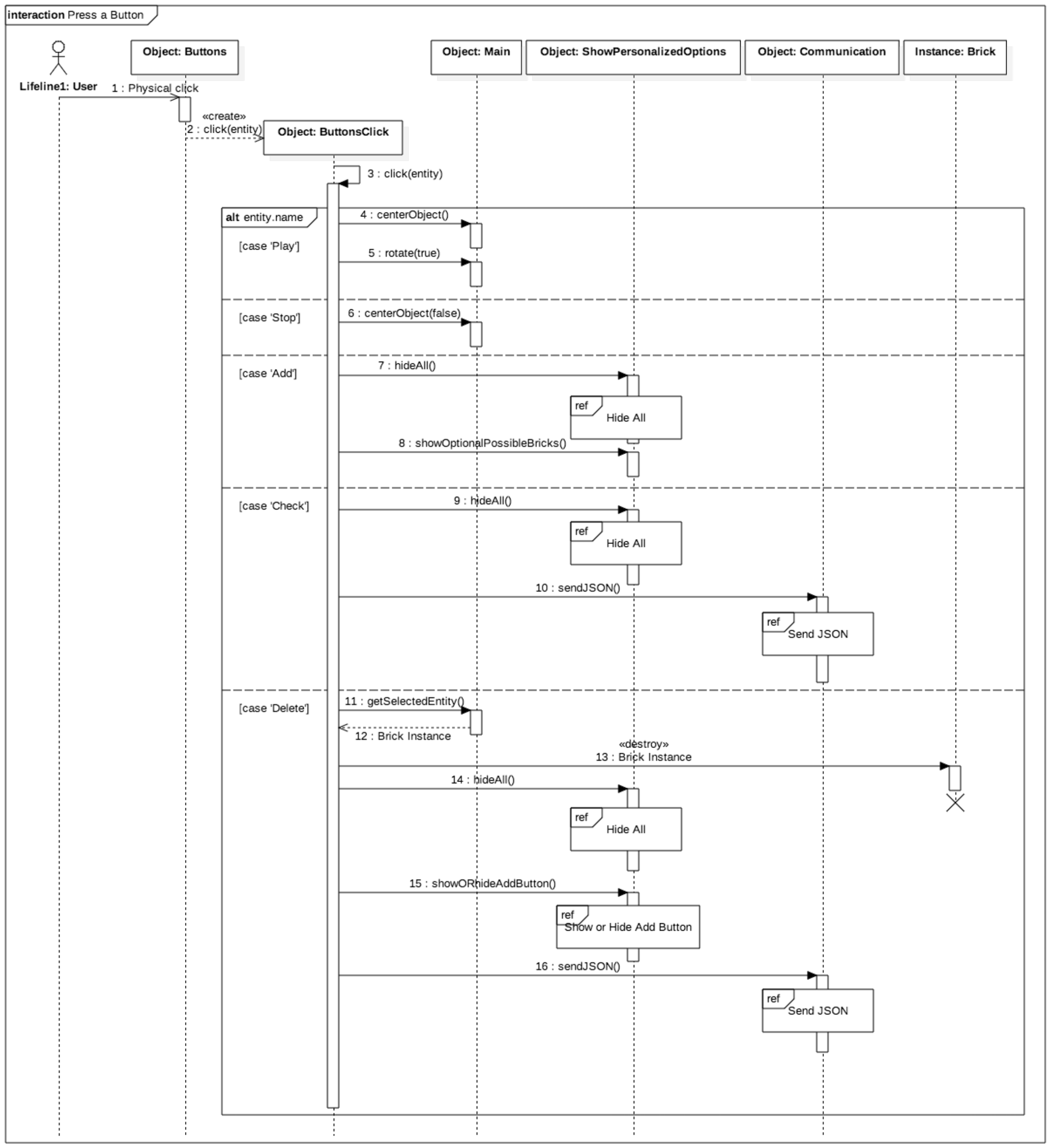


Figure 34 Sequence Diagram: Press a Button

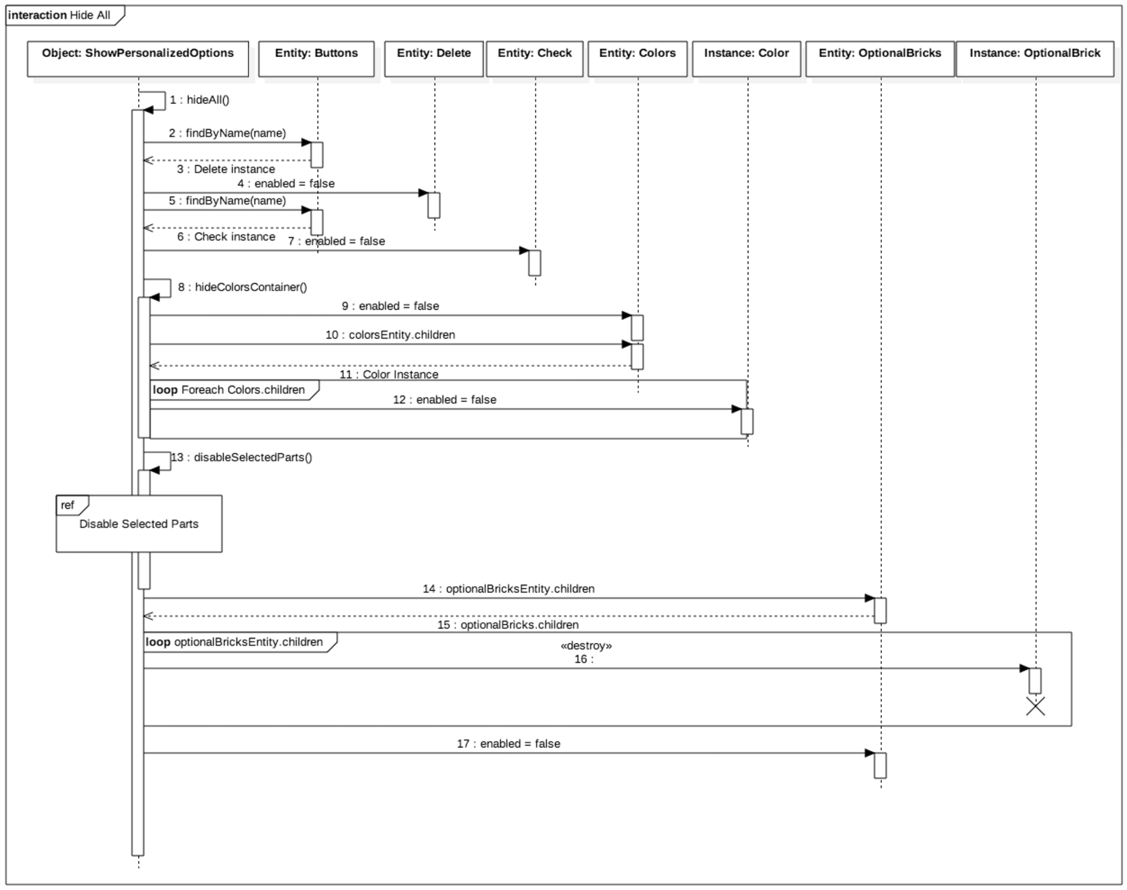


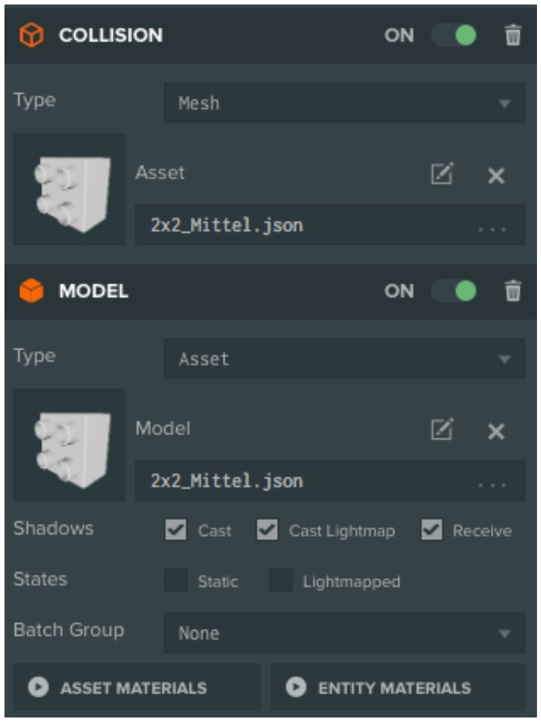
Figure 35 Sequence Diagram: Hide All

Press a Piece

When the user selects any piece of the Lego Duplo car, the system uses a *RayCast* to identify where exactly on the screen the touch was made. It casts a ray, from point origin, in a particular direction, of a specified length, against all colliders in the scene[2]. Hence, it is necessary for each piece to provide a collision element, as shown in Figure 36. The type of collision is *Mesh*, having the same CAD element of the model as an asset, insuring that if the touch is in any part of a piece, it can recognize which was the selected one.

Depending on the type of brick or of the chassis, the system carries out a series of procedures to show the "Color Picker" on the left side of the screen (like the one in Figure 37). These methods are described in the Sequence Diagram in Figure 39.

[2] *https://docs.unity3d.com/ScriptReference/Physics.Raycast.html*



COLLISION ON

Type: Mesh

Asset: 2x2_Mittel.json

MODEL ON

Type: Asset

Model: 2x2_Mittel.json

Shadows: [✓ Cast] [✓ Cast Lightmap] [✓ Receive]

States: Static, Lightmapped

Batch Group: None

Figure 36 Collision and Model Element to a piece of the type 2x2x2

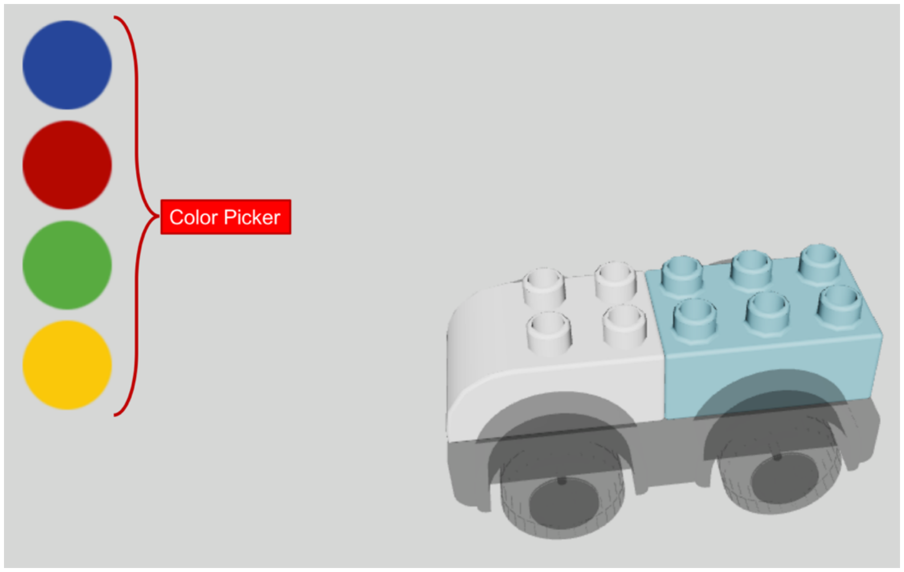


Figure 37 Color Picker on Screen

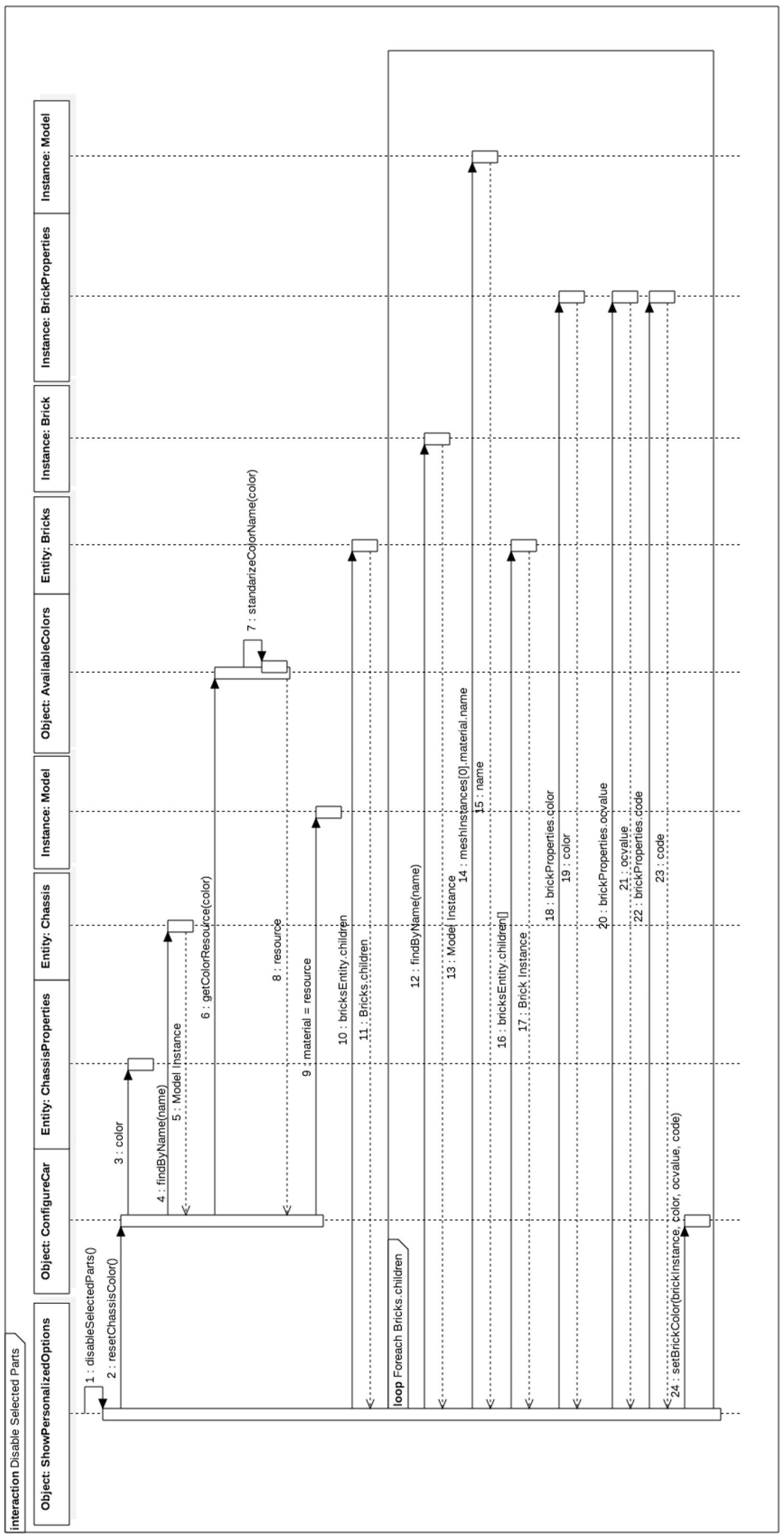

interaction Disable Selected Parts
Object: ShowPersonalizedOptions
Object: ConfigureCar
Entity: ChassisProperties
Entity: Chassis
Instance: Model
Object: AvailableColors
Entity: Bricks
Instance: Brick
Instance: BrickProperties
Instance: Model
1 : disableSelectedParts()
2 : resetChassisColor()
3 : color
4 : findByName(name)
5 : Model Instance
6 : getColorResource(color)
7 : standarizeColorName(color)
8 : resource
9 : material = resource
10 : bricksEntity.children
11 : Bricks.children
loop Foreach Bricks.children
12 : findByName(name)
13 : Model Instance
14 : meshInstances[0].material.name
15 : name
16 : bricksEntity.children[]
17 : Brick Instance
18 : brickProperties.color
19 : color
20 : brickProperties.ocvalue
21 : ocvalue
22 : brickProperties.code
23 : code
24 : setBrickColor(brickInstance, color, ocvalue, code)

Figure 38 Sequence Diagram: Disable Selected Parts

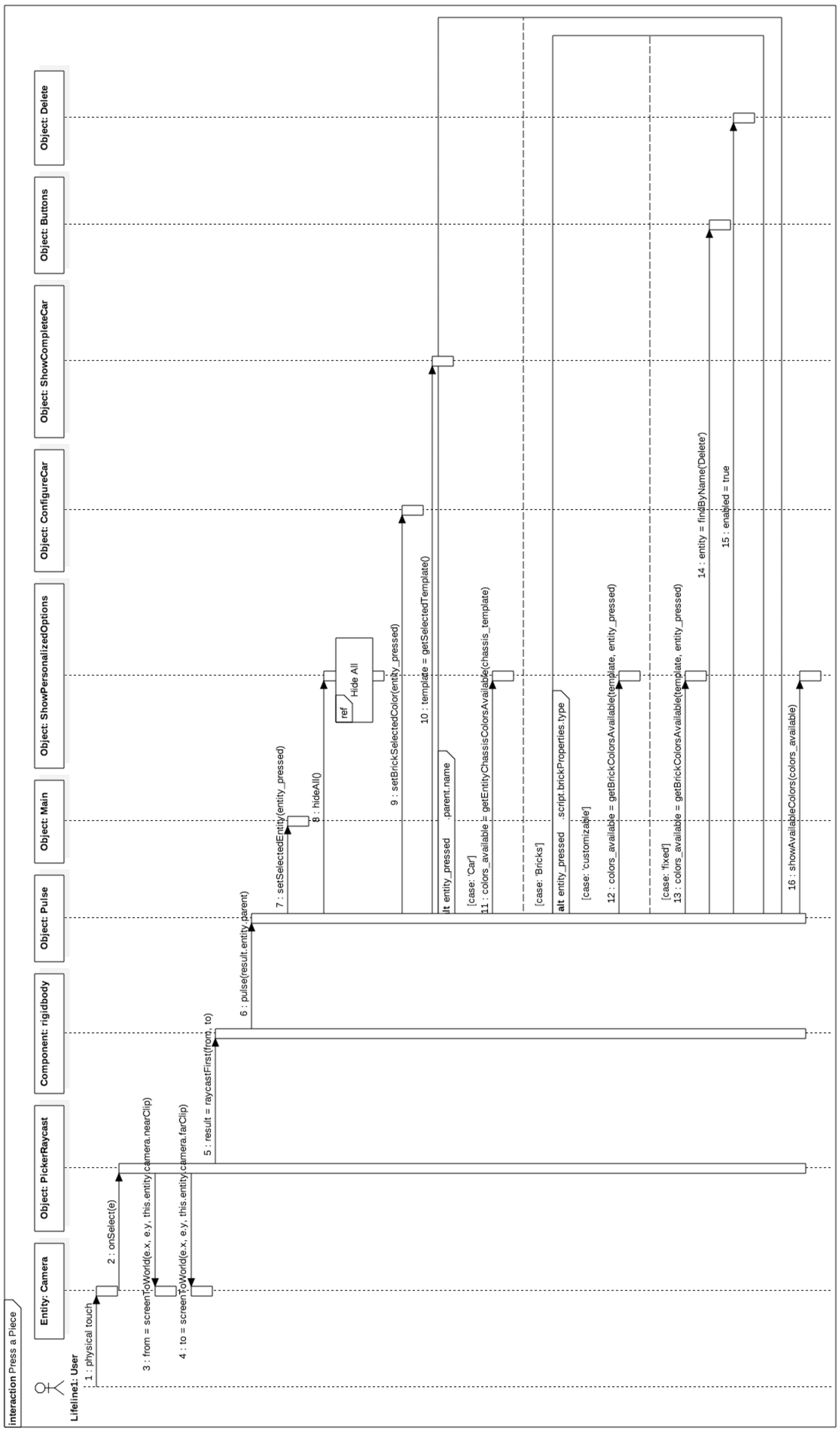

interaction Press a Piece
Lifeline1: User
Entity: Camera
Object: PickerRaycast
Component: rigidbody
Object: Pulse
Object: Main
Object: ShowPersonalizedOptions
Object: ConfigureCar
Object: ShowCompleteCar
Object: Buttons
Object: Delete
1 : physical touch
2 : onSelect(e)
3 : from = screenToWorld(e.x, e.y, this.entity.camera.nearClip)
4 : to = screenToWorld(e.x, e.y, this.entity.camera.farClip)
5 : result = raycastFirst(from, to)
6 : pulse(result.entity.parent)
7 : setSelectedEntity(entity_pressed)
8 : hideAll()
ref
Hide All
9 : setBrickSelectedColor(entity_pressed)
10 : template = getSelectedTemplate()
alt entity_pressed.parent.name
[case: 'Car']
11 : colors_available = getEntityChassisColorsAvailable(chassis_template)
[case: 'Bricks']
alt entity_pressed.script.brickProperties.type
[case: 'customizable']
12 : colors_available = getBrickColorsAvailable(template, entity_pressed)
[case: 'fixed']
13 : colors_available = getBrickColorsAvailable(template, entity_pressed)
14 : entity = findByName('Delete')
15 : enabled = true
16 : showAvailableColors(colors_available)

Figure 39 Sequence Diagram: Press a Piece

Press Optional Bricks Picker

As previously mentioned, after the “Add” button is pressed, the “Optional Bricks Picker” is displayed on the screen (Figure 32). This picker shows every piece contained in the *optional* array of the input JSON container. To display the specific bricks, the system clones each element from the entity *OptionalBricksList* to the entity *OptionalBricks* and this last entity is the one that can be enabled to appear on the screen.

When a piece has more than two different color possibilities, but the coordinates and brick name are the same, it is shown only one time in the picker. This new entity includes the script “optionalBricksClick”, that, when it is instanced, creates an event click listener that differentiates which new piece was pressed.

One of the features that the system provides is that if the new part is bigger than the previous one or overlaps with any other(s), it deletes the existing bricks from the car.

This last process is done in Sequence 6 in Figure 44 Sequence Diagram: Press the Optional Bricks Picker. The approach to finding out this overlapping is by iteratively assigning a volume to every brick with the position in the space and making comparisons among them. The code for this function is presented in Figure 41 and Figure 42, and is explained next.

Firstly, by using the brick name, it determines the width, length, and height of both bricks (the brick to add and the existent brick). The name of every brick starts with the size measured by the number of studs. It is in the format “widthxlengthxheight”, and it can include at the end the attribute “-bow” when the brick has a fillet in one of its top edges, “-bows” when the fillet is in both of them, or the name of a special attribute like in lines 21-26 in Table 2. Nevertheless, the width and the length is switched when the rotation in the ‘z’ axis changes the orientation, for example, for a rotation in the ‘z axis’ of an even number, the width, length and height are exactly like in Figure 40; however, for an odd number the width and length are switched. The values of these variables are assigned in lines 11-44 in Figure 41.

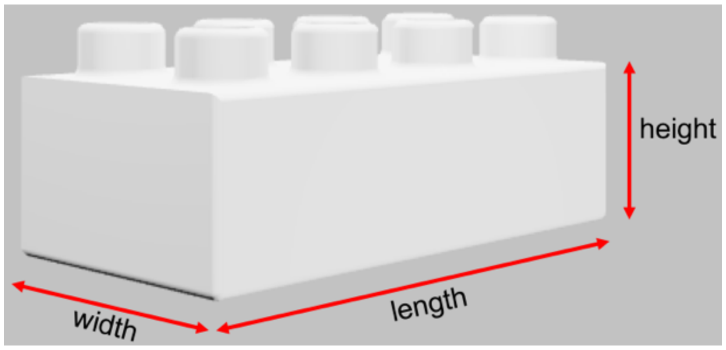


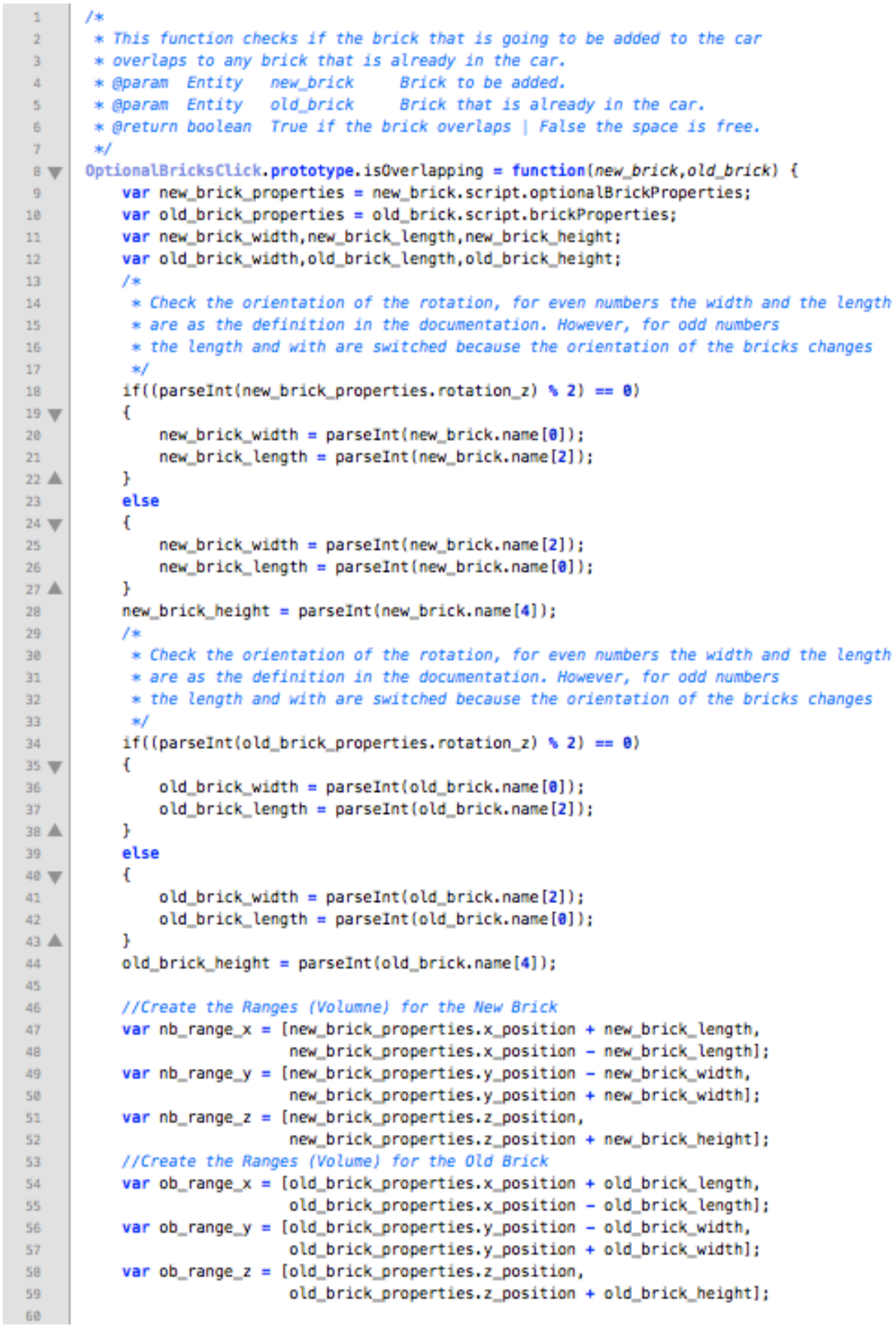
Figure 40 Stud dimensions

Secondly, the function assigns ranges or limits for both pieces in every axis. This assignment contemplates the coordinates that are described in Sub-Section Coordinates System, meaning that if one range is inside another, the pieces are overlapping. The procedure of calculating the ranges can be seen in lines 46-59 in Figure 41.

Lastly, it checks if one range is inside another. To do that is a complex procedure involving many logical operators. To determine that two pieces are colliding, there has to be an overlapping in the three axes and then the function returns *true*. These operations are done in lines 61-188 in Figure 42.

The procedure described before is done iteratively for every active brick in the car and every time the function described before *isOverlapping()* returns *true* and the type of brick to be deleted is of the type *customizable*, the sequences 8 and 9 in Figure 44 are executed to delete the brick also from the array *customizable* and adding it to the array *optional* in the car template. These two procedures are required in order to let the user add the brick again if it is needed.

Nevertheless, not minding if the brick is *customizable* or *optional*, it is deleted from the car so that the new brick can fit in the position defined on the template.

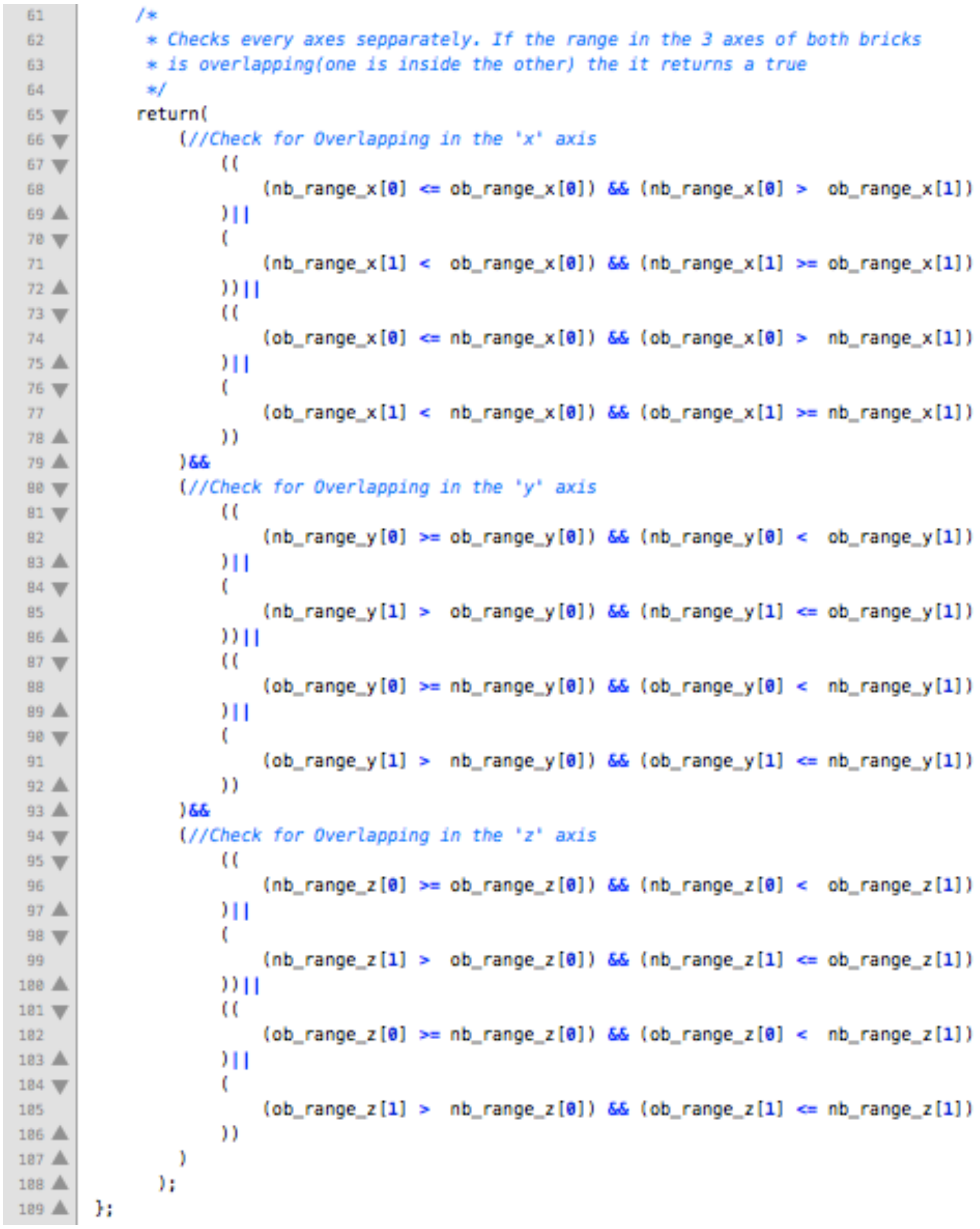


```
/*
 * This function checks if the brick that is going to be added to the car
 * overlaps to any brick that is already in the car.
 * @param  Entity   new_brick     Brick to be added.
 * @param  Entity   old_brick     Brick that is already in the car.
 * @return boolean  True if the brick overlaps | False the space is free.
 */
OptionalBricksClick.prototype.isOverlapping = function(new_brick,old_brick) {
    var new_brick_properties = new_brick.script.optionalBrickProperties;
    var old_brick_properties = old_brick.script.brickProperties;
    var new_brick_width,new_brick_length,new_brick_height;
    var old_brick_width,old_brick_length,old_brick_height;
    /*
     * Check the orientation of the rotation, for even numbers the width and the length
     * are as the definition in the documentation. However, for odd numbers
     * the length and with are switched because the orientation of the bricks changes
     */
    if((parseInt(new_brick_properties.rotation_z) % 2) == 0)
    {
        new_brick_width = parseInt(new_brick.name[0]);
        new_brick_length = parseInt(new_brick.name[2]);
    }
    else
    {
        new_brick_width = parseInt(new_brick.name[2]);
        new_brick_length = parseInt(new_brick.name[0]);
    }
    new_brick_height = parseInt(new_brick.name[4]);
    /*
     * Check the orientation of the rotation, for even numbers the width and the length
     * are as the definition in the documentation. However, for odd numbers
     * the length and with are switched because the orientation of the bricks changes
     */
    if((parseInt(old_brick_properties.rotation_z) % 2) == 0)
    {
        old_brick_width = parseInt(old_brick.name[0]);
        old_brick_length = parseInt(old_brick.name[2]);
    }
    else
    {
        old_brick_width = parseInt(old_brick.name[2]);
        old_brick_length = parseInt(old_brick.name[0]);
    }
    old_brick_height = parseInt(old_brick.name[4]);

    //Create the Ranges (Volumne) for the New Brick
    var nb_range_x = [new_brick_properties.x_position + new_brick_length,
                      new_brick_properties.x_position - new_brick_length];
    var nb_range_y = [new_brick_properties.y_position - new_brick_width,
                      new_brick_properties.y_position + new_brick_width];
    var nb_range_z = [new_brick_properties.z_position,
                      new_brick_properties.z_position + new_brick_height];
    //Create the Ranges (Volume) for the Old Brick
    var ob_range_x = [old_brick_properties.x_position + old_brick_length,
                      old_brick_properties.x_position - old_brick_length];
    var ob_range_y = [old_brick_properties.y_position - old_brick_width,
                      old_brick_properties.y_position + old_brick_width];
    var ob_range_z = [old_brick_properties.z_position,
                      old_brick_properties.z_position + old_brick_height];

```

Figure 41 Function isOverlapping() part-1



```
    /*
     * Checks every axes sepparately. If the range in the 3 axes of both bricks
     * is overlapping(one is inside the other) the it returns a true
     */
    return(
        (//Check for Overlapping in the 'x' axis
            ((
                (nb_range_x[0] <= ob_range_x[0]) && (nb_range_x[0] >  ob_range_x[1])
            )||
            (
                (nb_range_x[1] <  ob_range_x[0]) && (nb_range_x[1] >= ob_range_x[1])
            ))||
            ((
                (ob_range_x[0] <= nb_range_x[0]) && (ob_range_x[0] >  nb_range_x[1])
            )||
            (
                (ob_range_x[1] <  nb_range_x[0]) && (ob_range_x[1] >= nb_range_x[1])
            ))
        )&&
        (//Check for Overlapping in the 'y' axis
            ((
                (nb_range_y[0] >= ob_range_y[0]) && (nb_range_y[0] <  ob_range_y[1])
            )||
            (
                (nb_range_y[1] >  ob_range_y[0]) && (nb_range_y[1] <= ob_range_y[1])
            ))||
            ((
                (ob_range_y[0] >= nb_range_y[0]) && (ob_range_y[0] <  nb_range_y[1])
            )||
            (
                (ob_range_y[1] >  nb_range_y[0]) && (ob_range_y[1] <= nb_range_y[1])
            ))
        )&&
        (//Check for Overlapping in the 'z' axis
            ((
                (nb_range_z[0] >= ob_range_z[0]) && (nb_range_z[0] <  ob_range_z[1])
            )||
            (
                (nb_range_z[1] >  ob_range_z[0]) && (nb_range_z[1] <= ob_range_z[1])
            ))||
            ((
                (ob_range_z[0] >= nb_range_z[0]) && (ob_range_z[0] <  nb_range_z[1])
            )||
            (
                (ob_range_z[1] >  nb_range_z[0]) && (ob_range_z[1] <= nb_range_z[1])
            ))
        )
      );
};
```

Figure 42 Function isOverlapping() part-2

Press Colors Picker

This process is very similar to the previous one. The distinction is that it clones from the entity *ColorsList* to *Colors*, and this last entity has a script "colorsClick" for the event click listener, but the basic operation is the same. e.g., in Figure 43 at first, the car is displayed. The user clicks on the chassis, making it transparent and showing the “Colors Picker”. Finally, the red color inside the “Colors Picker” is selected, changing the color of the chassis to red consequently. This last process is described in the Sequence Diagram on Figure 45. Using also the following sub-sequences:

Figure 24 Sequence Diagram: Set Chassis Color, Figure 26 Sequence Diagram: Set Brick Color, and Figure 28 Sequence Diagram: Send JSON.

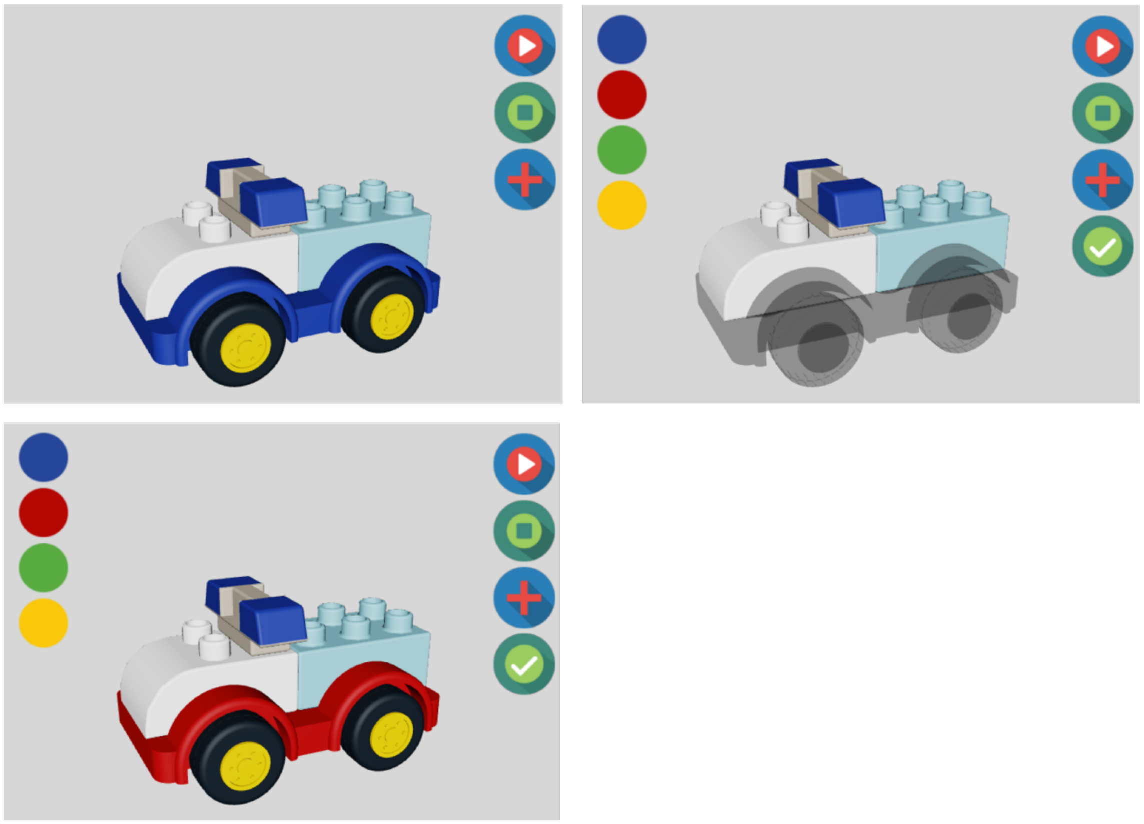


Figure 43 Sequence for changing the Chassis' Color

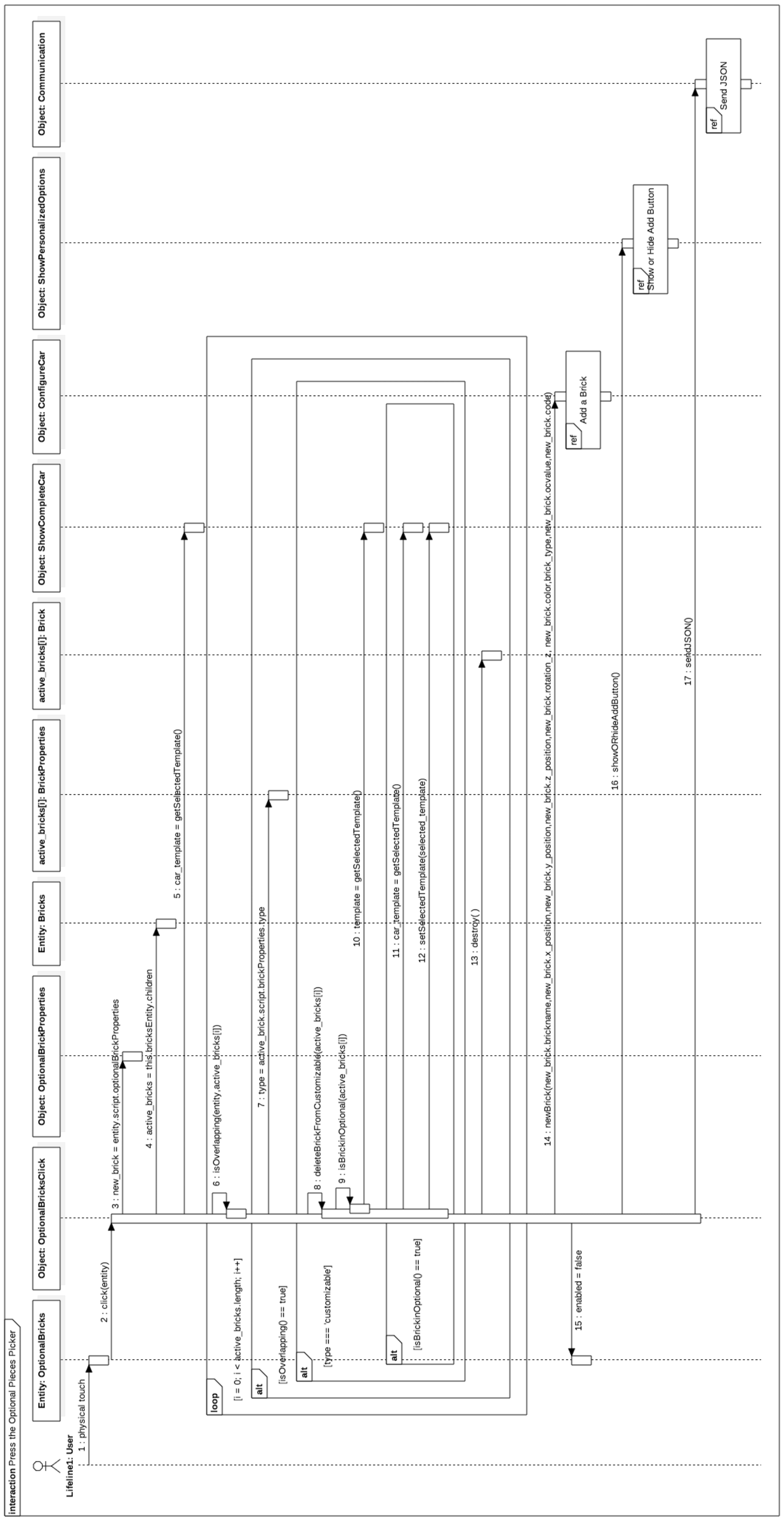

interaction Press the Optional Pieces Picker
Lifeline1: User
Entity: OptionalBricks
Object: OptionalBricksClick
Object: OptionalBrickProperties
Entity: Bricks
active_bricks[i]: BrickProperties
active_bricks[i]: Brick
Object: ShowCompleteCar
Object: ConfigureCar
Object: ShowPersonalizedOptions
Object: Communication
1 : physical touch
2 : click(entity)
3 : new_brick = entity.script.optionalBrickProperties
4 : active_bricks = this.bricksEntity.children
5 : car_template = getSelectedTemplate()
loop
[i = 0; i < active_bricks.length; i++]
6 : isOverlapping(entity,active_bricks[i])
alt
[isOverlapping() == true]
7 : type = active_brick.script.brickProperties.type
alt
[type === 'customizable']
8 : deleteBrickFromCustomizable(active_bricks[i])
9 : isBrickinOptional(active_bricks[i])
10 : template = getSelectedTemplate()
alt
[isBrickinOptional() == true]
11 : car_template = getSelectedTemplate()
12 : setSelectedTemplate(selected_template)
13 : destroy( )
14 : newBrick(new_brick.brickname,new_brick.x_position,new_brick.y_position,new_brick.z_position,new_brick.rotation_z, new_brick.color,brick_type,new_brick.ocvalue,new_brick.code)
ref
Add a Brick
15 : enabled = false
16 : showORhideAddButton()
ref
Show or Hide Add Button
17 : sendJSON()
ref
Send JSON

Figure 44 Sequence Diagram: Press the Optional Bricks Picker

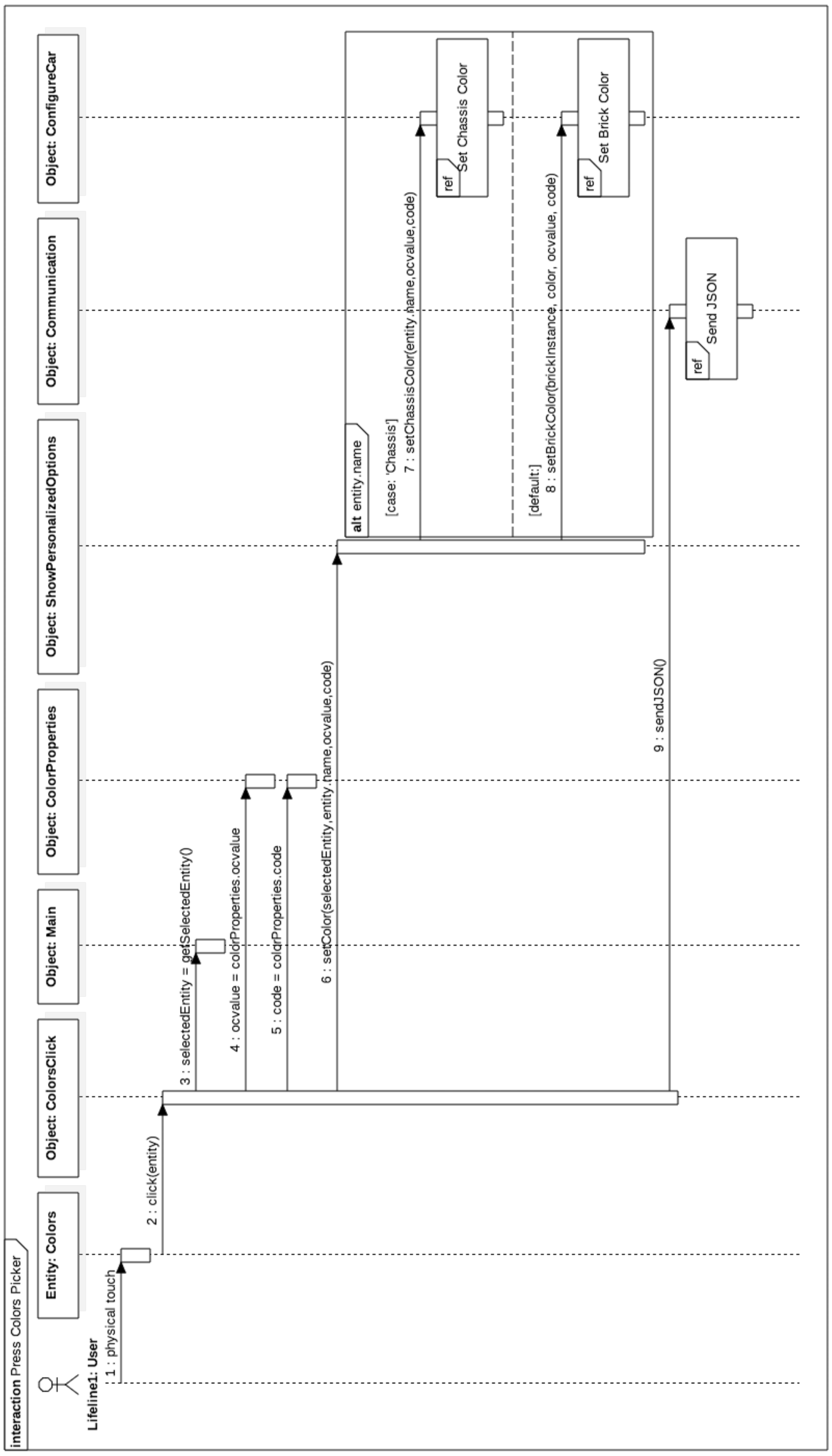

interaction Press Colors Picker
Lifeline1: User
Entity: Colors
Object: ColorsClick
Object: Main
Object: ColorProperties
Object: ShowPersonalizedOptions
Object: Communication
Object: ConfigureCar
1 : physical touch
2 : click(entity)
3 : selectedEntity = getSelectedEntity()
4 : ocvalue = colorProperties.ocvalue
5 : code = colorProperties.code
6 : setColor(selectedEntity,entity.name,ocvalue,code)
alt entity.name
[case: 'Chassis']
7 : setChassisColor(entity,name,ocvalue,code)
ref
Set Chassis Color
[default:]
8 : setBrickColor(brickInstance, color, ocvalue, code)
ref
Set Brick Color
9 : sendJSON()
ref
Send JSON

Figure 45 Sequence Diagram: Press the Colors Picker

System 4 - IoT Adapter

As mentioned in the previous sections, the administrator of the system is in charge of updating the definition of both bricks and templates (Lego Duplo car), so that they are comprehensible for the translator and subsequently for the Viewer. This data is stored in the database *mes* under the tables: *factory_itemgroup*, *factory_item* and *opencart_actions*. The relationships of this tables can be seen in Figure 46.

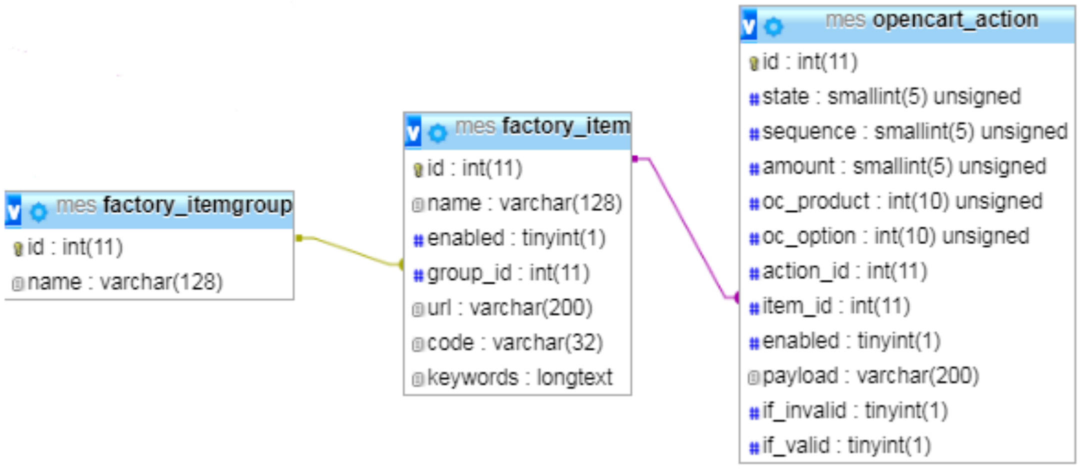
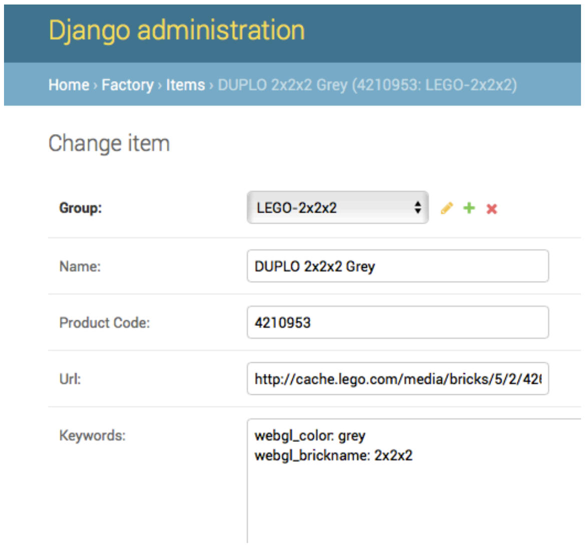


Figure 46 Extract of the Entity-Relation Diagram for the database "mes"

To edit the information in the database, the IoT Adapter presents a “Django Administration” tool, that groups the information and allows changes to the data. To give the appropriate description of a brick, each item must have the appropriate *KEYWORDS* definition; inside the keywords field, two variables in a YAML format are contained. These variables are:

- **webgl_brickname**: This property determines which piece is displayed. There are 26 possibilities for this and is important that the written name coincides with any of the brick names from the table in Annexure 5 in the column “Brick Name for Template Definition”.
- **webgl_color**: Every piece in the catalog can be displayed with any of the 30 possibilities for the color shown in Annexure 4. In the case of special pieces, like emergency lights, engine, dumper, etc., the color can be changed as well, although some parts of the piece have a default color already. The color name has to be the same as any from the column “Color Name”; nevertheless, the name is not case sensitive.

Figure 47 presents an example of the Django administration tool when editing the information of a brick. In this case the brick is a "2x2x2 Lego Duplo (Row #2 in Table 2 Available Bricks)" with the color "grey" (Row #15 in Table 1 Available Colors).


Django administration
Home › Factory › Items › DUPLO 2x2x2 Grey (4210953: LEGO-2x2x2)
Change item

Group:	LEGO-2x2x2
Name:	DUPLO 2x2x2 Grey
Product Code:	4210953
Url:	http://cache.lego.com/media/bricks/5/2/42
Keywords:	webgl_color: grey webgl_brickname: 2x2x2

Figure 47 Keywords Configuration of a Brick

After every brick is configured with the correct *keywords*, every product (Lego Duplo car) has to be added into the table *opencart_action*. In here, the important values are the *state*, *oc_product*, *oc_option* and *payload*. Figure 48 gives an example of the values that the product "Roadster" has. The values are further explained:

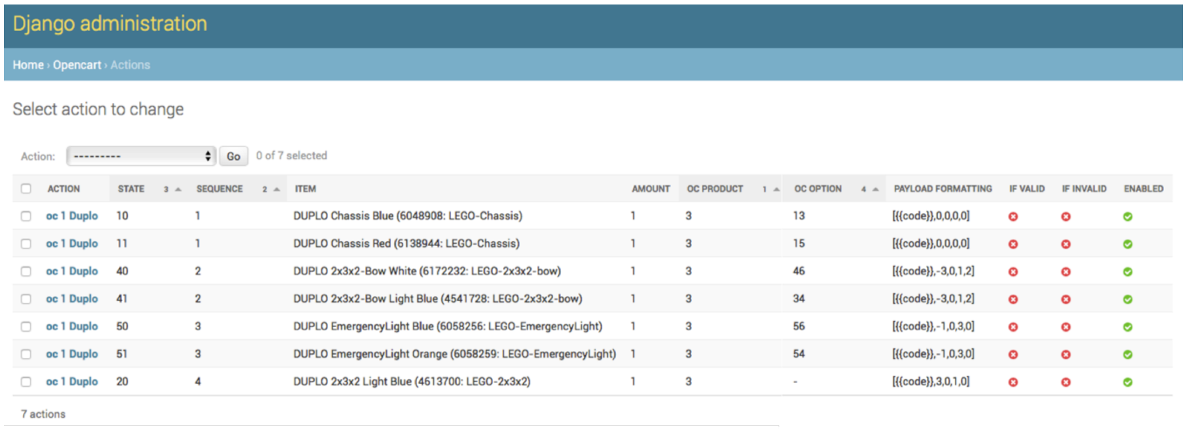
- **state:** The state property is a number that associates the different brick or color options for the same position. All the chassis should have a value from 119, the rest of the pieces have a value of a multiple of 10 starting with 20. There is no limit for the amount of pieces, but the possibilities for each position is limited to 10 (e.g., 229).
- **oc_product:** This is the id for each product (Lego Duplo car) inside OpenCart. It is important that the value in the database is exactly the same as the one in OpenCart since this is the way the system associates which template to display. This value is commonly displayed in the *url* of the product using the *GET* method, for example, the *url* for the product "Roadster" is: *http://localhost/OpenCart/index.php?route=product/product&path=2&product_id=3* thus, the id for this product is three.
- **oc_option:** This number is the value that OpenCart uses to reference a radio button to every element that can be added or deleted from a car. In Annexure 7 there is a guide on

how to obtain this value. As a common rule, every piece always has a different value; however, this number can be null, meaning that the piece is fixed.

- **payload:** This string contains the coordinates of a singular piece. The format is:

  [{code}x_coordinate,y_coordinate,z_coordinate, rotation_z]

  In the next section, an explanation for each coordinate is given.



Django administration

Home › Opencart › Actions

Select action to change

Action: --------- Go 0 of 7 selected

ACTION	STATE 3 ▲	SEQUENCE 2 ▲	ITEM	AMOUNT	OC PRODUCT 1 ▲	OC OPTION 4 ▲	PAYLOAD FORMATTING	IF VALID	IF INVALID	ENABLED
oc 1 Duplo	10	1	DUPLO Chassis Blue (6048908: LEGO-Chassis)	1	3	13	[{{code}},0,0,0,0]	✗	✗	✓
oc 1 Duplo	11	1	DUPLO Chassis Red (6138944: LEGO-Chassis)	1	3	15	[{{code}},0,0,0,0]	✗	✗	✓
oc 1 Duplo	40	2	DUPLO 2x3x2-Bow White (6172232: LEGO-2x3x2-bow)	1	3	46	[{{code}},-3,0,1,2]	✗	✗	✓
oc 1 Duplo	41	2	DUPLO 2x3x2-Bow Light Blue (4541728: LEGO-2x3x2-bow)	1	3	34	[{{code}},-3,0,1,2]	✗	✗	✓
oc 1 Duplo	50	3	DUPLO EmergencyLight Blue (6058256: LEGO-EmergencyLight)	1	3	56	[{{code}},-1,0,3,0]	✗	✗	✓
oc 1 Duplo	51	3	DUPLO EmergencyLight Orange (6058259: LEGO-EmergencyLight)	1	3	54	[{{code}},-1,0,3,0]	✗	✗	✓
oc 1 Duplo	20	4	DUPLO 2x3x2 Light Blue (4613700: LEGO-2x3x2)	1	3	-	[{{code}},3,0,1,0]	✗	✗	✓

7 actions

Figure 48 Database example values for the product roadster in the table opencart_actions

Coordinates System

To simplify the localization of any brick in the space, the Viewer and the table *opencart_action* use the same three-dimensional cartesian coordinate system; the origin 0 and the orientation of the axes are shown in Figure 50. The center of a piece is shown in Figure 49; The value for the coordinates indicates the number of units translated from this center point to any of the axes.

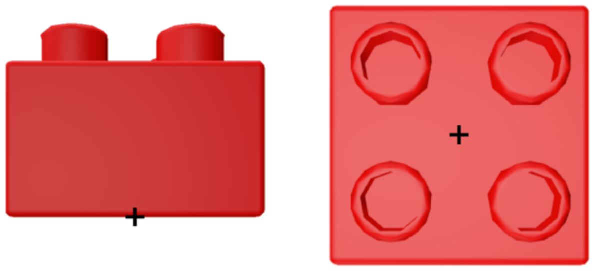


Figure 49 Center of a piece

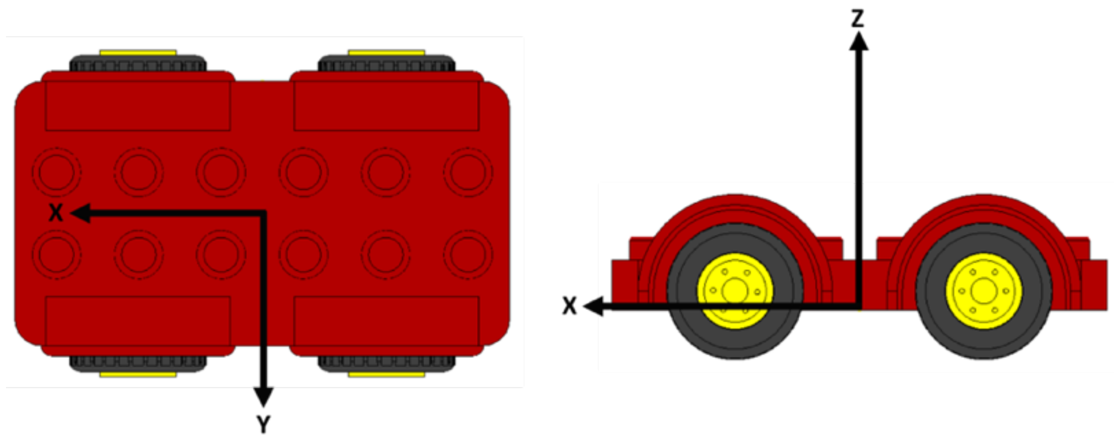


Figure 50 Three-dimensional Cartesian coordinate system and orientation of the axes

Figure 51 shows an example of how a piece can be located on the x-axis. This axis is the one that moves along the chassis. Figure 51 a) illustrates the axis orientation for the top view of the car. In the case of Figure 51 b) , the coordinate in the x-axis is 0 because the piece is located on the origin of the chassis for this particular axis.

Figure 51 c) indicates a translation in this axis of -1. This case is not physically possible, since the studs of both the chassis and the brick does not match; therefore, the administrator is the one responsible for the correct alignment of the blocks; nevertheless, the system allows to virtually place them in any part of the space. Finally, Figure 51 d) shows a 2x2x2 cube with an coordinate of -2. As a general reference, to move any piece in the x-axis one stud, the number in the x-position has to be incremented by 2.

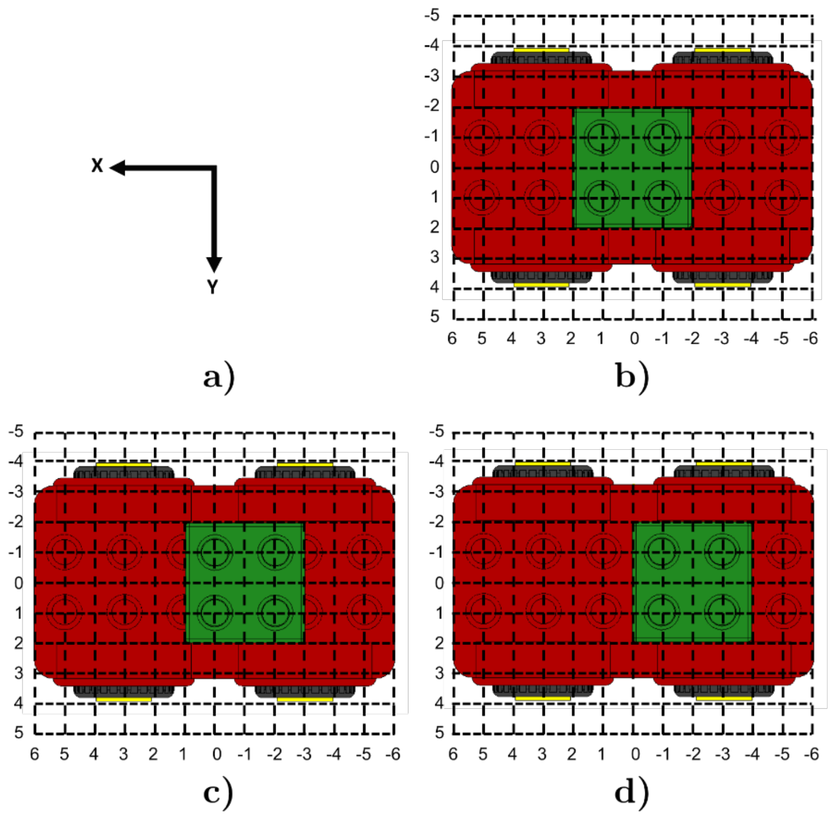


Figure 51 Example of translation values on the x-axis

The localization of bricks in the y-axis is similar to the one for the x-axis. Figure 52 shows an example of how a piece can be positioned on the y-axis. Figure 52 a) illustrates the axes orientation for the top view of the car. Figure 52 b), describes a value for the y-axis of 0. In this particular axis, 0 is the only value where a piece stays inside the chassis.

Figure 52 c) indicates a y-translation of -1. This translation is not physically possible since the stud of both the chassis and the brick does not match and there is a collision between the brick and the chassis bumper. Finally, Figure 52 d) shows a 2x2x2 cube with a y-coordinate of -2. As a general reference, to move any piece in the y-axis one stud, just as in the x-axis, the number in y_position has to be incremented by 2.

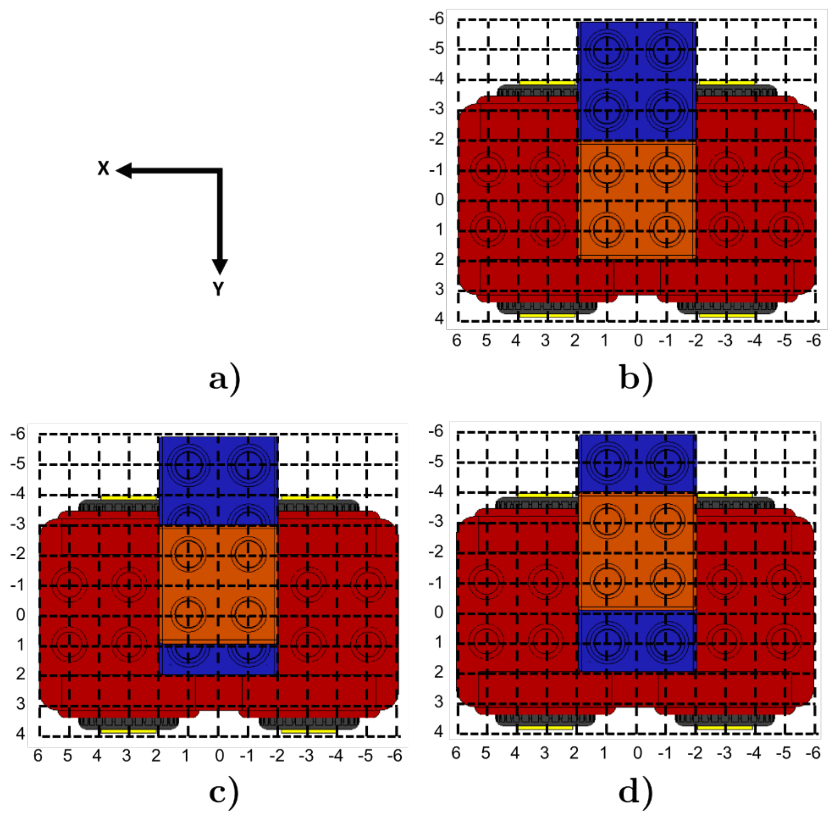


Figure 52 Example of translation values on the y-axis

Last, the z-axis defines the translation in the vertical position. The value used for the location in this axis is given by the height of the pieces. The chassis is the only one with a z-position value of 0. Every brick located on the surface of the chassis has a value of 1, like the one shown in Figure 53 b). Figure 53 c) has a z-position value of 3 since the brick in Figure 53 b) has a height of 2. This new piece (the orange one) has an altitude of 1, that is the smallest possible height. Consequently, for Figure 53 d), the value of the position in z is 4.

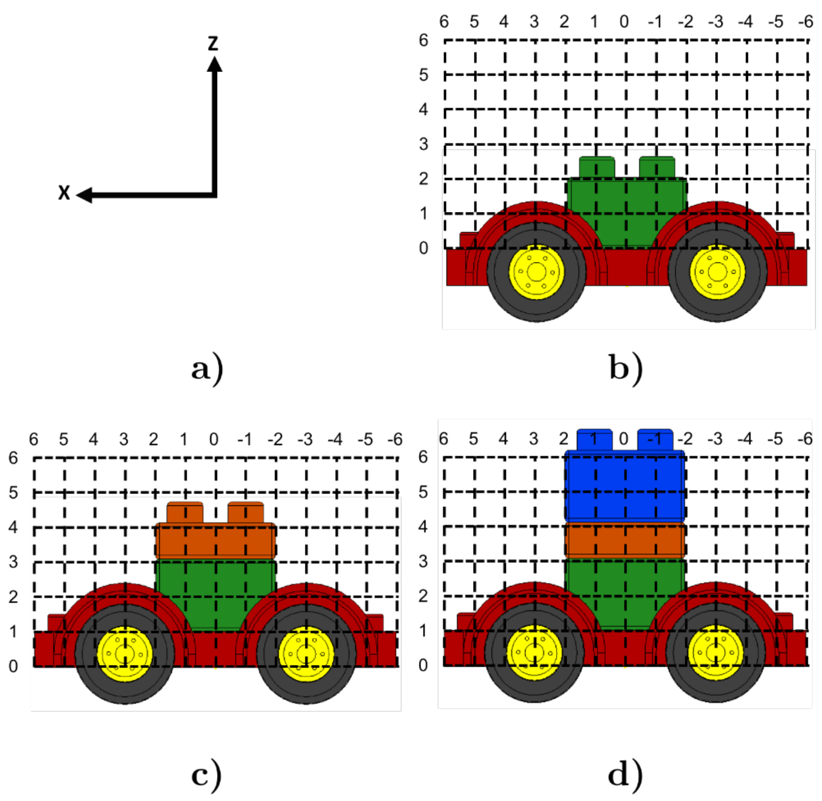


Figure 53 Example of translation values on the z-axis

Another property that it is included in the localization of the pieces is a rotation in the z-axis. In here, the value is an integer from 0 to 4, that represents increments of 90 degrees. When the rotation value is set to zero, if the part has a fillet in one of its edges, this roundness will be oriented to the left side of the chassis (where 'x' is positive). The, each value increments the rotation by 90 degrees going in a counter-clockwise orientation as shown in Figure 54 a). Figure 54 b) is a piece of the type 2x2x2-bow, and as previously mentioned, since the rotation in the z-axis is 0, the fillet is pointing to the left. Figure 54 c) and Figure 54 d) have a rotation of 2 and 3 respectively.

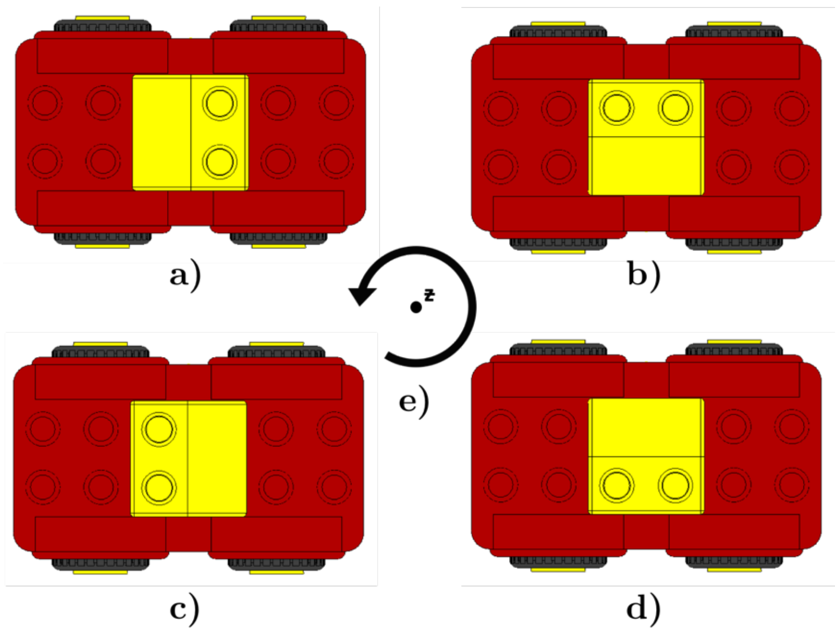


Figure 54 Example of rotation values on the z-axis

After the payload is formed using the three coordinates plus the rotation described before, the system has enough information for the REST API Server, so when an action is consulted, the data contained is compatible with the Viewer.

Results

In some projects the desired results are different to the actual results. In the previous section it was described how is the system implemented and how it should work on a normal operation. However, in order to validate that the expected results are correct, the planned testing can come in handy.

Testing is crucial during the software development cycle because it does not only reduce possible notorious errors; it also increases the performance of any system, allows lower maintenance costs and improves the user experience. There are several methods to do testing. One important strategy is the white-box, which permits to examine the internal structure of any program, testing the program's logic [26].

To do the testing, several cases are proposed from the beginning, having the intention of making a thorough inspection. The authors G. J. Myers, C. Sandler, and T. Badgett [26] present ten principles for the planning of assessments, all of them of vital significance, e.g., Principle five stipulates that "Test cases must be written for input conditions that are invalid and unexpected, as well as for those that are valid and expected".

The previous principle intends finding conditions where the software is more likely to crash because just as the author summarizes, a good test case is one that has a high probability of detecting an undiscovered error [26].

This chapter presents six test cases, all of them consider the ten principles specified by G. J. Myers, C. Sandler, and T. Badgett [26] and are carefully designed to test absolutely every process of the software, both in communication among systems and just as a Viewer. At first, they give a summary about the test or explain the intention of the test. Then the date and time as well as the prerequisites to make the test possible to operate are elaborated.

Moreover, the test procedure defines with high accuracy all the steps to execute. In the section “Expected Results”, the expected output is given once the steps are performed. Finally, the last section offers remarks, comments, and pieces of advice for future work, in case some error was not handled.

Test Case 1

Summary	**Testing the visualization of every car product.**
Date and Time of the test(s)	First test: January 26th, 2018. 18:54 hrs. Second test: January 29th, 2018. 14:25 hrs.
Prerequisites	1. The Product (in OpenCart) has a defined action in the database “mes” in the table opencart_action. 2. The IoT Adapter and the Web Server are running. 3. The OpenCart modification is installed. 4. The 3DViewer layout is properly configured. 5. Every product has the layout “3DViewer”.
Test Procedure	1. Select the language: English. 2. Open OpenCart. 3. Open the “Cars Selector”. 4. Iteratively select every product: 4.1. Roadster. 4.2. Sedan. 4.3. Station Wagon. 4.4. Truck. 5. Verify that the car is appropriately displayed. 6. Change the language to German. 7. Iteratively select every Product: 7.1. Sportwagen. 7.2. Kombi. 7.3. Limousine. 7.4. LKW. 8. Verify that the car is appropriately displayed. 9. Create a product (Car) in OpenCart. 10. Enable the layout 3DViewer for this product. 11. Do not add the template to the table opencart_action.

Test Procedure	12. Wait for the error message.
Expected Results	1. Each product, when English is enabled, is displayed correctly. 2. Each product, when German is enabled, is displayed correctly. 3. For the product that does not have a defined action in the database, the page does not disable the loading screen and shows an error message.
Actual Results	1. For English, every product was displayed correctly. 2. For German, every product was displayed correctly. 3. The page disabled the loading screen and never showed the error message. 4. For the second test, the page kept the loading screen and never showed the error message.
Remarks and Future Work	1. When a product does not have an action associated with the database, it is recommended to use the default layout instead the 3DViewer. 2. This wrong result was corrected and tested again; now when a product does not have an action associated with the database, it keeps the loading screen enabled and shows an error message.

Table 1 Test Case 1

Test Case 2

Summary	**Testing the correct association between each piece and the OC Value.**
Date and Time of the test	First test: January 26th, 2018. 17:05 hrs. Second test: January 28th, 2018. 18:34 hrs. Third test: January 29th, 2018. 13:52 hrs. Fourth test: January 29th, 2018. 14:22 hrs.

Prerequisites	1. Every product has a defined action in the database "mes" in the table opencart_action. 2. The radio button value in OpenCart of every option has the same value as the one in the column oc_option in the table opencart_action in the database mes. 3. The IoT Adapter and the Web Server running.
Test Procedure	1. Customize every product in any language with all the possibilities, and for every customization possibility, add the car to the "Shopping Cart". 2. Verify that the car was successfully added. 3. For every time a product is added to the queue, open the "Shopping Cart" and verify that the product name, the quantity and the options are the correct ones.
Expected Results	1. It is possible to customize every car and add it with the modifications to the "Shopping Cart". 2. Every time a modification is made in the Viewer, the changes can be reflected in OpenCart. 3. When a product is added to the "Shopping Cart", it displays with text all the modifications and the extra pieces if they were added.
Actual Results	1. Every modification was added correctly to the "Shopping Cart". 2. When the "V-Engine" was added to the "Station Wagon", the position was incorrect. The coordinates in the payload were corrected and tested again. The result was still incorrect. 3. A mistake was found inside PlayCanvas, the engine was misaligned so a correction was made. 4. The test was executed again with a successful output this time.

Remarks and Future Work	1. It is crucial to do this test every time a product is created in order to verify the correct association with the radio buttons values and the proper definition of the coordinates. 2. When an item is removed from the car in the Viewer, it does not update the changes in OpenCart. The reason is that if a radio button is activated, it cannot be deactivated again. 3. If there is an option described inside an action, but in OpenCart it is not activated, the system shows it anyway, like the "Petrol Tank" for the product "Truck". The solution is not including pieces in the response JSON from the API if they are not available for the car.

Table 2 Test Case 2

Test Case 3

Summary	**Verifying that every brick and every color displays correctly.**
Date and Time of the test(s)	First test: January 27th, 2018. 20:00 hrs. Second test: January 27th, 2018. 20:33 hrs. Third test: January 27th, 2018. 20:52 hrs. Fourth test: January 27th, 2018. 21:27 hrs.
Prerequisites	1. The Viewer is used in exhibition mode (the variable *"EXHIBITION_MODE"* in the script *"Main.js"* is set to *"true"*. 2. The Viewer is connected neither to OpenCart nor the IoT Adapter.
Test Procedure	1. Replace the content inside the asset 10790245 (input_json_example) to a new JSON that matches the following requirements: 1.1. The value of the *"json_return_type"* is 4. 1.2. The array "chassis" is empty. 1.3. The array "fixed" contains every normal brick (Lines 1 – 20 in Table 2 Available Bricks). Every one of the previous

Test Procedure	bricks has a different color, and the value of its attribute state is a multiple of 10. 1.4. The array "customizable" contains some extra random pieces that in total have all the available colors in Table 1 Available Colors (every piece has to be added every time for each color). The value of its attribute state is a multiple of 10 for every different type of piece. 1.5. Every brick in points 1.2 and 1.3 has different coordinates in the 3 axes and is not overlapping with any other piece. 1.6. The array "optional" is filled with all the pieces in Table 2 Available Bricks. The coordinates of this parts can overlap with any other brick(s). 2. Run the application in any web browser with the console open (Developer Tools). 3. Verify that every brick is displayed in the correct position and with the right color. 4. Select one of the customizable pieces and change the color to every possibility. Every time verify that the JSON that is printed in the console is updating the changes. 5. Add all the optional bricks. Verify that all the pieces with whom they are overlapping are deleted from the Product and the JSON in the console.
Expected Results	1. Every piece is displayed correctly in the defined coordinates. 2. All the colors are shown correctly. 3. When a customizable piece is pressed, the "Colors Picker" shows the correct options, and after selecting one color, the piece updates its color to the new one. 4. When the button "Add" is pressed, the "Optional Bricks Picker" is displayed with the correct options. After a piece is selected, it is added to the product inside the specified coordinates. Then, if the new piece is overlapping an existing brick, the old brick is erased

Expected Results	from the car and from the JSON, because this last one contains only the active bricks.
Actual Results	1. Every brick was displayed in the correct position except for the 2x3x1. There was a misalignment in PlayCanvas, but the error was solved before testing again. 2. All the colors were shown correctly. 3. Every element of the color picker was displayed; however, the “lightbrown” had the incorrect image and the “mediumblue” was not working. Both errors were corrected and retested. 4. When detecting the overlapping, the system was working incorrectly when the pieces were rotated, because there was a mistake getting the range of the bricks. After some modifications, this error was eliminated and after the last test, no further bugs where encountered.
Remarks and Future Work	1. For future work, it is necessary to have better collision management for irregular bricks. So far it works fine when pieces are overlapping; nonetheless, the special of pieces (e.g., ladder) does not detect their surroundings completely, particularly in the irregular borders.

Table 3 Test Case 3

Test Case 4

Summary	**Using the OpenCart implementation and the IoT Adapter in different servers**
Date and Time of the test(s)	First test: January 28th, 2018. 17:35 hrs. Second test: January 29th, 2018. 14:11 hrs.
Prerequisites	1. The OpenCart implementation is running on the WebServer.

Prerequisites	2. The OpenCart modification is installed and the Layout “3DViewer” is created and associated with a particular product (e.g. Roadster)
Test Procedure	1. Change the location of The IoT Adapter (the one containing the database) to a different server (with a different IP address) inside the same LAN. 2. The variable *“api”* in the file *“/3DPlugin/translator.js”* has the value “’/api/oc_action/’”. Change this value to *“’/<new_server_ip_address>*/api/oc_action/*’”*. 3. Open any product that has the layout 3DViewer enabled. 4. Verify that the product is displayed. 5. Change the location of the IoT Adapter (the one containing the database) to a different server (with a different IP address) but outside the LAN. 6. Repeat steps 2-4. 7. Return the variable to its original value.
Expected Results	The product is displayed even if they are in different servers and the error message is not shown.
Actual Results	The communication in both systems in different servers did not work.
Remarks and Future Work	For both tests (inside and outside the LAN), it only works when the REST API Server contains the header accepting connections from other IP addresses, e.g., *"Access-Control-Allow-Origin: <opencart_impementation_ip_address>"*.

Table 4 Test Case 4

Test Case 5

Summary	**Interrupting the connection with the database**
Date and Time of the test(s)	Test: January 28th, 2018. 18:14 hrs.
Prerequisites	1. The OpenCart implementation and the IoT Adapter are running in different servers.

Prerequisites	2. The OpenCart modification is installed and the Layout "3DViewer" is created and associated to a particular product (e.g. Roadster).
Test Procedure	1. Open any product that has the layout 3DViewer enabled. 2. Interrupt the connection with the IoT Adapter (database).
Expected Results	1. If there is no connection with the database, the system tries to connect after 10, 15, 20 and 30 seconds. Each time it is not able to connect it shows the error message in the console: "[Error] Error loading after x seconds, the system will try to load again". After 30 seconds, it shows the error message in the console and on the screen: "The system could not load the template properly. Please try again later or contact the administrator." 2. The loading screen is never disabled.
Actual Results	The system worked as expected.
Remarks and Future Work	When the loading screen is not being disabled, a recommendation is to open the console in the web browser to look for errors.

Table 5 Test Case 5

Test Case 6

Summary	**Using inexistent bricks, colors or an incorrect JSON format**
Date and Time of the test(s)	First test: January 28th, 2018. 18:56 hrs. Second test: January 28th, 2018. 17:14 hrs. Third test: January 28th, 2018. 19:29 hrs. Fourth test: January 28th, 2018. 19:41 hrs.
Prerequisites	1. A product (in OpenCart) has a defined action in the database "mes" in the table opencart_action. 2. The IoT Adapter and the Web Server are running. 3. The OpenCart modification is installed.

Prerequisites	4. The 3DVIewer layout is properly configured. 5. At least one product has the layout "3DViewer".
Test Procedure **Test Procedure**	1. Edit the "brickname" attribute in keywords in the database for a particular brick that is contained in a car (e.g., LEGO-2x3x2 Lightblue) and change it to a non-existing piece (e.g., 2x6x2). 2. Open a car that contains that piece (e.g., Roadster). 3. Edit the "brickname" attribute in keywords in the database for a particular special brick that is contained in a car (e.g., LEGO-EmergencyLight Blue) and make an intentional typo mistake (e.g., 2x1x2-emerg**a**ncylight). 4. Open a car that contains that piece (e.g., Roadster) and press the Add button. 5. Edit the "color" attribute in keywords in the database for a particular brick that is contained in a car (e.g., LEGO-2x3x2 Lightblue) and change it to a non-existing color (e.g., silver). 6. Open a car that contains that piece (e.g., Roadster). 7. Edit the "color" attribute in keywords in the database for a particular brick that is contained in a car (e.g., LEGO-2x3x2 Lightblue) and change it to a non-existing variation of a color (e.g., dusty red). 8. Open a car that contains that piece (e.g., Roadster).
Expected Results	1. For points 1-2. If the piece does not exist, when the system is trying to display the car, it shows an error message on the screen and on the console, specifying which brick does not exist. 2. For points 3-4. If the special piece does not exist, the system displays the car with no problems, but when the user presses the Add button, it shows an error message in the screen and on the console, specifying which brick does not exist. 3. For points 5-6, when a color does not exist, the system shows the piece in the default color black.

Expected Results	4. For points 7-8, when a variation of color does not exist, it looks for a color with a similar name (e.g., for “dusty red” it uses the color “red”).
Actual Results	1. The error message was only printed in the web browser’s console. 2. When pressing the “Add” button, the system was crashing. After a second modification, now when the color does not exist, it prints the error: “The color: dusty red is not on the list of possible colors. Instead, it will be displayed with the color: red. Please see the documentation for a full list of all the color possibilities”. 3. The system displayed the brick with the default color; however, there is no error message that informs the reason to the user. After a modification, now when the color does not exist, it prints the error: “The color: xxxxx is not on the list of possible colors. Instead it will be displayed with the color: xxxxx. Please see the documentation for a full list of all the color possibilities”. 4. The system displayed the brick with the similar color; however, there is no error message that informs the reason to the user. As point 3, now it prints an error.
Remarks and Future Work	1. As a remark, it is important to always open the web browser’s console when something is not working as expected, because many errors are reported. 2. As a future work, the loading screen should be disabled only when all the introduced bricks exist.

Conclusions

One critical part of this project was the testing. As it was mentioned before, one of the goals of the thesis was to produce software that is not only reliable, but that also holds remarkably high quality. To be able to ensure the previous statement, the Section 4. Test Cases has to be reviewed.

After the execution of the six test cases, at least one error in five out of six cases was detected. Aforementioned does not imply that the software was poorly designed, on the contrary, the inspections were so meticulous that all errors that could happen for particular circumstances have been detected. This allows to suppress all of them before the last release.

Moreover, two types of flaws were identified. One type of mistakes was the result of mistakes in the code. The other type of mistakes was a result of misguides of the user when inputting data. The first type entailed that some functions were revised and edited to prevent the errors from happening and were tested repetitively until it was certain that the bug was eradicated.

The second type of bugs cannot be completely eliminated, since it cannot be assured better that the user or even the administrator will not introduce incorrect data. Nevertheless, thanks to identifying in which situations this could possibly happen, some algorithms were designed to give feedback to the user when an error is encountered, always providing sufficient information to understand the reason and how can it be reversed.

Another key aspect was to provide software that is not only understandable for a machine, but is also comprehensive for humans; meaning that the functions and code involved is designed in a certain way that is readable. This point was achieved firstly by the naming the variables and functions. Instead of using just random characters or short and incomprehensible acronyms, all the names are extensive and with a full relationship with their respective purpose.

Secondly, the class description is something to take into consideration when establishing that the software design is comprehensible. Not only there is a full description of the purpose of them, but there also is enough information of every single function with its respective parameters and return values, enabling that updates of the code are implemented entirely intuitional.

Third and last for this point, the so-mentioned functions have another peculiarity. Every single one of them is involved in at least one Sequence Diagram, so that it is not only possible to assume what they do, but also how they communicate among each other, which is their respective sequence and when exactly they are called with which information.

Another critical point that should be considered is the adoption of a methodology. By using the incremental process model, the system adapted to the constantly changing requirements through the development process. The first delivery and the last version, despite the fact that they both were applications for the same purpose, in the end, the interfaces were completely different, and it is possible to affirm that the last one is more attached to the user needs. Moreover, by testing the code in each deployment iteration, several bugs were found and solved along the way, this brought many benefits while executing the tests at the later stages of the project because the number of bugs was considerably low.

The aspects just mentioned describe the software and its functions from the perspective of the administrator and future developers. When looking at the software from the point of view of the final user, the following points are to be mentioned: The Viewer provides an evident smoothness when a person is using it. Since the first time it was opened by a third-person, the conception of the behavior was natural and presented no challenges for the operation. Aforesaid is translated as intuitivity because it is not only easy to use, it also increases the satisfaction of the customer providing a better experience in the purchasing process.

Future Work

The main challenge when further developing the software is to couple it better with mobile devices. Further development of the software resides in the better coupling with mobile devices. The software is fully optimized for multi-touch screens and portable devices such as smartphones and tablets, providing different gestures for zooming and movement of the product; however, there is the need to export it and adapt it in a native application. It is proven that this increases the satisfaction of the user and can reduce the loading time.

Furthermore, the inclusion of even more customized products is unavoidable. So far it is based on predefined templates, but it is undoubtedly scalable to present a complete customization experience that enables users to create products from scratch by establishing some limitations at first, but then providing more flexibility.

The last point of improvement comes in the use of specific entities for particular types of bricks. If the further versions of PlayCanvas let the user to modify the model asset via code, all of the specific entities can be eliminated and a generic entity can be used. Furthermore, the CAD model can be received as a parameter inside the JSON container,   with this alteration it could be possible to support not only Lego Duplo bricks, but any CAD model.

References

[1] Siemens, “Industrie 4.0 - The Fourth Industrial Revolution,” 2013.

[2] S. Wang, J. Wan, D. Li, and C. Zhang, “Implementing Smart Factory of Industrie 4.0: An Outlook,” *Int. J. Distrib. Sens. Networks*, vol. 12, no. 1, p. 3159805, 2016.

[3] H. Kagermann, J. Helbig, A. Hellinger, and W. Wahlster, *Recommendations for Implementing the Strategic Initiative INDUSTRIE 4.0: Securing the Future of German Manufacturing Industry ; Final Report of the Industrie 4.0 Working Group*. Forschungsunion, 2013.

[4] C. Stary and M. Neubauer, “Industrial Challenges,” in *S-BPM in the Production Industry: A Stakeholder Approach*, M. Neubauer and C. Stary, Eds. Cham: Springer International Publishing, 2017, pp. 7–25.

[5] B. Pressman, Roger; Maxim, *Software Engineering. A practitioner’s approach*, 8th editio. New York: McGraw Hill, 2015.

[6] P. Mcdermid, J; Rook, “The Software Engineers Reference Book,” *Software Development Process Models*, pp. 15–28, 1993.

[7] E. Hozdić, “Smart factory for industry 4.0: A review,” *Int. J. Mod. Manuf. Technol.*, vol. 2, no. 1, pp. 2067–3604, 2015.

[8] F. Wortmann and K. Flüchter, “Internet of things. Technology and Value Added,” *Bus. Inf. Syst. Eng.*, vol. 57, no. 3, pp. 221–224, 2015.

[9] L. Atzori, A. Iera, and G. Morabito, “The internet of things: A survey,” *Comput. networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[10] D. Giusto, “A. lera, G. Morabito, l. Atzori (Eds.) The Internet of Things.” Springer, 2010.

[11] “OpenCart - Business Shopping Cart Software.” [Online]. Available: https://certifiedhosting.com/opencart-hosting/. [Accessed: 29-Jan-2018].

[12] “5 Benefits That Make OpenCart Rock.” [Online]. Available: https://www.shopping-cart-migration.com/blog/46-opencart/12797-5-benefits-that-make-opencart-rock. [Accessed: 29-Jan-2018].

[13] “What Is OpenCart? OpenCart Advantages | OpenCart Store Manager, Product Manager for OpenCart.” [Online]. Available: https://www.opencartmanager.com/useful-articles/general-information-on-opencart/what-is-opencart-opencart-advantages/. [Accessed: 29-Jan-2018].

[14] “Modifications - OpenCart Documentation.” [Online]. Available: http://docs.opencart.com/extension/modifications/. [Accessed: 29-Jan-2018].

[15] Khronos Group, “OpenGL Overview,” *Opengl.Org*, 2012. [Online]. Available: https://www.opengl.org/about/. [Accessed: 29-Jan-2018].

[16] M. Woo and O. A. R. Board, *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.2*. Addison-Wesley, 1999.

[17] Khronos Group, “WebGL Overview - The Khronos Group Inc.” .

[18] T. Parisi, *WebGL: Up and Running: Building 3D Graphics for the Web*. O’Reilly Media, 2012.

[19] A. Mazur, C. Mills, and Ladybenko, “Building up a basic demo with PlayCanvas - Game.” [Online]. Available: https://developer.mozilla.org/en-US/docs/Games/Techniques/3D_on_the_web/Building_up_a_basic_demo_with_PlayCanvas. [Accessed: 31-Jan-2018].

[20] Michael Larabel, “PlayCanvas Browser-Based Game Engine Open-Sourced - Phoronix.” [Online]. Available: https://www.phoronix.com/scan.php?page=news_item&px=MTcxMDA. [Accessed: 31-Jan-2018].

[21] PlayCanvas Ltd, “PlayCanvas Engine.” [Online]. Available: https://github.com/playcanvas/engine. [Accessed: 3Jan-2018].

[22] M. & Tools, “StarUML - Open Source UML Tool.” .

[23] “StarUML.” .

[24] “FBX | Adaptable File Formats for 3D Animation Software | Autodesk.” [Online]. Available: https://www.autodesk.com/products/fbx/overview. [Accessed: 04-Feb-2018].

[25] “Autodesk - Autodesk FBX - FBX® 2013.3 Converter.” [Online]. Available: http://usa.autodesk.com/adsk/servlet/pc/item?siteID=123112&id=22694909. [Accessed: 04-Feb-2018].

[26] G. J. Myers, C. Sandler, and T. Badgett, *The Art of Software Testing*. Wiley, 2011.

Annexures

In this section there are presented some of important information for the whole comprehension of this thesis. At the end of the document there is a DVD containing some files such as CAD models and the code source.

Annexure 1

Thesis Proposal Outline

FH Aachen & Center for Engineering and Industrial Development
Thesis Proposal Outline

Name of Student: Jose Gerardo Gomez Mendez Student Id: 3097036

Areas of interest and Justification of Research/Project

Areas of Interest

- Programming
- IoT
- Industry 4.0

Justification of Research/Project

The Industry 4.0 Lab in the FH Aachen, provides the tools and mechanisms to interconnect different assembly cells, so synergistically, they carry out the assembling process of customized Lego Duplo Cars.

The current area of opportunity in this system is the customization process. In Industry 4.0, it is essential to support customized demands by the customer; hence the involving of the final customer with the assembly process is fundamental. Nonetheless, to let the user customize it owns products so they can be suitable to build, the infrastructure of the system should provide the necessary means to customize the outcome within the restrictions of the assembling process.

Furthermore, another critical point inside Industry 4.0 is the scalability and flexibility that any system should provide, this delivers a considerable responsibility when developing software for this purpose. The only method to provide a system with the necessary means to be scalable resides in a correct use of the Software's Engineering.

This last point always represents a challenge to the designers of the program because not only the software has to work well; instead, it has to be completely reliable, bug-free and with meaningful documentation that makes possible for future engineers to understand the code and adapt it to the new functionalities that eventually will attain.

Moreover, the usage of IT tools, such as PCs, tablets, and smartphones, opens an area of opportunity for the interaction of the user with the system. Most devices used nowadays possess enough characteristics to support features like 3D visualization, rendering of complex images, and high bandwidth connectivity. All these points make feasible but also crucial, the integration of the upcoming technologies with any process inside Industry 4.0.

Research/Project Goals and Objectives

Research/Project Goals

Objective

To design and develop a 3D web application for the customization of predefined Lego Duplo Cars, and implementing it inside an already operating Industry 4.0 system.

Specific Objectives

1. To explore the different platforms for the development of portable, scalable and robust web-applications.
2. To create an application in the chosen platform that can receive the template of any Lego Duplo Car Template as an initial parameter and provide the tools for customizing it and transmit the modifications back to the original sender.
3. To develop an intermediate program that interprets the data from the database, where all the templates are stored, and translate this data into a standard format understandable by the Viewer (JSON).
4. To adapt both programs inside an OpenCart Plugin, so that they can be integrated into the running Industry 4.0 platform.
5. To provide all the documentation for the installation and maintenance of the system.

Research/Project Objectives

This Thesis pretends to be an integration of the customization process inside major IT devices, specially oriented to the touch-displays present in tablets and smartphones.

It is intended to use as well some of the techniques for the good practice of software development. One of this practices is the delivering of vast documentation, mostly based in UML 2.0 that makes the system completely clear to understand. The second point comes in the reliability, taking a considerable attention in the testing process, so the system is examined against all possible failures, all of this described in the Test Cases so the future administrator of the system knows contingency methods against some of the crashes that may occur.

The last point resides on the flexibility. This project aches to have a flexible and scalable system, therefore providing the tools to adapt it to distinct requirements, making it possible to adjust it to the late technologies and giving the plausibility to use it for any purpose different than Lego Duplo Cars, as long as it is build using Lego Duplo Bricks.

Summary of Preliminary Literature Review

[1] R. Pressman and B. Maxim, Software Engineering. A practitioner's approach, 8th edition ed., New York: McGraw Hill, 2015.

[2] J. Mcdermid and P. Rook, "Software Development Process Models," *The Software Engineers Reference Book,* pp. 15-28, 1993.

Proposed institution, proposed technical advisor and methodology

Institution	FH Aachen
Name of Technical Advisor	Prof. Dr.-Ing. Jörg Wollert
Methodology *(brief description of how the goals and objectives are going to be achieved)*	
The adopted methodology on the development of the software was the Incremental Process Model. The important on choosing a specific process model is that they provide stability, control, and organization to an activity that can, if left uncontrolled, become quite chaotic [1]. When an incremental model is used, the first increment is often a core product. That is, basic requirements are addressed but many supplementary features (some known, others unknown) remain undelivered. The core product is used by the customer (or undergoes detailed evaluation). As a result of use and/ or evaluation, a plan is developed for the next increment. The plan addresses the modification of the core product to better meet the needs of the customer and the delivery of additional features and functionality. This process is repeated following the delivery of each increment, until the complete product is produced. [1]. The incremental model applies linear sequences in a staggered fashion as calendar time progresses. Each linear sequence produces deliverable "increments" of the software [2]. Figure 1 presents an example of this process model in a graph of project calendar time against the software functionality and features. The main advantage in this process model is that all the features, functionalities and even the scope of the system can be refined and adapted to the changes in the other systems. Regarding the quality of the software, the possibility of testing the software in each iteration of development reduces the number of bugs considerably and prevents the appearance of significant errors that can take most of the developing time.	

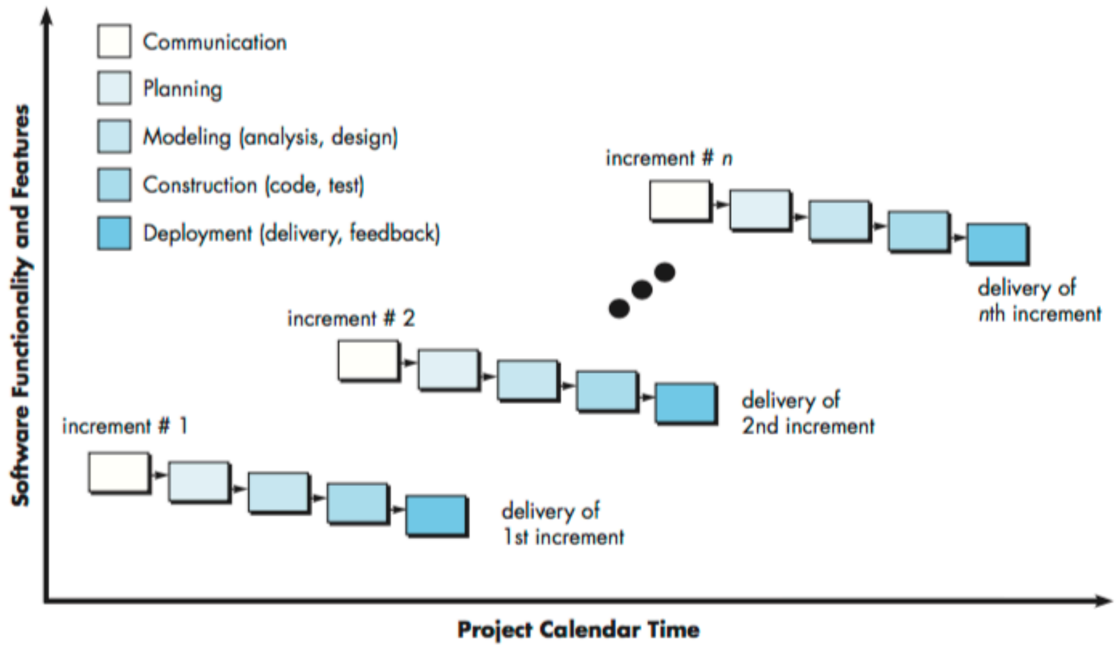

Communication
Planning
Modeling (analysis, design)
Construction (code, test)
Deployment (delivery, feedback)
Software Functionality and Features
increment # 1
delivery of 1st increment
increment # 2
delivery of 2nd increment
increment # n
delivery of nth increment
Project Calendar Time

Figure 55 Incremental Process Model, calendar time progress [1]

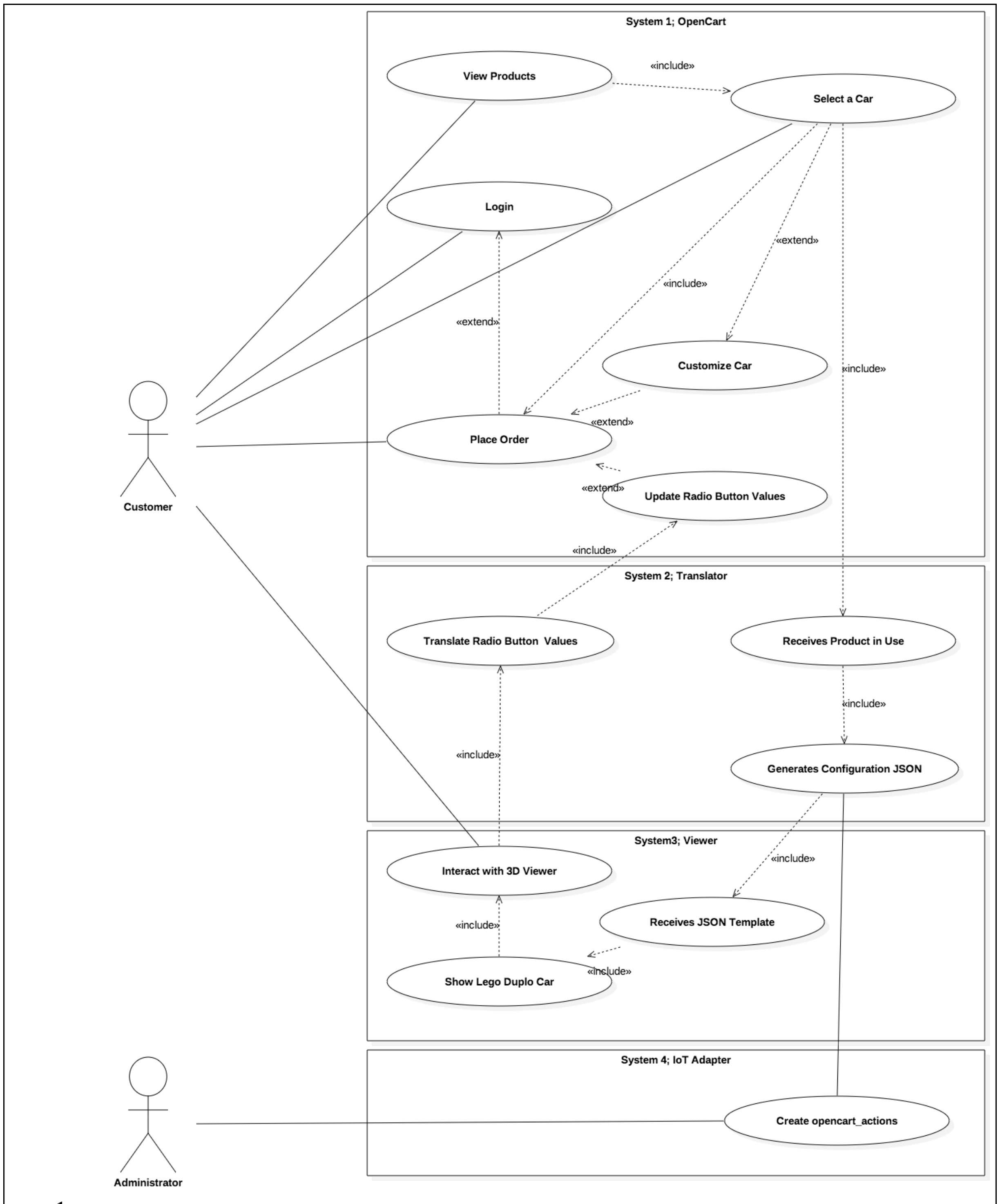
Anticipated Results

All the Systems that are included in this project are shown in the previous use case diagram.

The Systems 2 and 3 were developed from scratch and are already up and running.

Systems 1 and 4 were already in use in the industry 4.0 Lab but were enhanced for this project to be compatible with the Viewer.

Right now every system was tested and is compatible with the last version of the I4.0 implementation.


System 1; OpenCart
View Products
«include»
Select a Car
Login
«extend»
«include»
«extend»
Customize Car
«include»
«extend»
Place Order
«extend»
Update Radio Button Values
«include»
Customer
System 2; Translator
Translate Radio Button Values
Receives Product in Use
«include»
«include»
Generates Configuration JSON
System3; Viewer
Interact with 3D Viewer
«include»
Receives JSON Template
«include»
Show Lego Duplo Car
«include»
System 4; IoT Adapter
Create opencart_actions
Administrator

1.

Timeline

Activity	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Preliminary phase	■	■																							
Development			■	■	■	■	■	■	■	■	■	■	■	■	■	■									
Deployment												■	■	■	■	■									
Testing															■	■									
Writing of The Thesis																	■	■	■	■	■	■	■	■	
Send The Thesis For a Review																						■	■	■	
Present it at CIDESI																									■

Week 1 starts on: 01/09/2018

Possible Academic Advisors

Name of Advisor	Institution
Prof. Dr.-Ing. Jörg Wollert	FH-Aachen
	CIDESI

Annexure 2

Zulassung zur Master-Abschlussarbeit

FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

FH Aachen| Prüfungssekretariat FB8| Goethestraße 1| 52064 Aachen

Herrn
José Gerardo Gómez-Méndez
Zimmer 2319
Schillerstrasse 86
52064 Aachen

Zulassung zur Master-Abschlussarbeit

Sehr geehrter Herr Gómez-Méndez,

auf Ihren Antrag vom 20.10.2017 lasse ich Sie zur Abschlussarbeit zu. Das Thema und den Betreuer Ihrer Abschlussarbeit entnehmen Sie bitte der von Ihnen eingereichten Unterlage im Anhang.

Der Abgabetermin ist der 09.03.2018!

Der früheste Abgabetermin ist, unter Berücksichtigung des Postweges, der 26.01.2018!

Bis zu diesem Zeitpunkt legen Sie die Abschlussarbeit bitte in dreifacher Ausfertigung im Prüfungssekretariat des Fachbereichs Maschinenbau und Mechatronik, Goethestraße 1, vor. Eine Kurzfassung in digitaler Form übergeben Sie bitte Ihrem Prüfer.

Ein Exemplar der vorlegten Abschlussarbeit erhalten Sie sofort mit Siegel und Datum zurück. Dieses Exemplar halten Sie fünf Jahre für die FH Aachen vor, was Sie mit der Aufbewahrungspflicht dokumentieren.

Die Abschlussarbeit muss die schriftliche und unterschriebene Zusicherung enthalten, dass sie von Ihnen selbständig angefertigt wurde und keine anderen Quellen benutzt wurden, als die von Ihnen angegebenen.

Es sind folgende Prüfungsleistungen/ Unterlagen bis zum Kolloquium zu erbringen:

- **83143 Mechatronik Projekt**
- **Entlastungsbescheinigung**

Für die Zulassung zum Kolloquium vereinbaren Sie nach Abgabe Ihrer Abschlussarbeit einen Termin mit Ihrem Prüfer.

Mit freundlichen Grüßen

[illegible]

Prof. Dr.-Ing. Manfred Enning

Anlage: Thema der Abschlussarbeit

FH Aachen
Goethestraße 1
52064 Aachen
www.fh-aachen.de

Prüfungsausschuss
Prof. Dr.-Ing. Manfred Enning

Fachbereich 8
Maschinenbau und Mechatronik

Auskunft erteilt
G. Meuter-Donay
G. Zivkovic
T +49. 241. 6009 52321
F +49. 241. 6009 52681

pruefungsamt.maschbau@fh-aachen.de

Datum
20.10.2017

09.03

Einzureichen beim Prüfungssekretariat des Fachbereichs Maschinenbau und Mechatronik 26.01.

FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Antrag auf Zulassung zur Abschlussarbeit

Bachelorarbeit [ ] Masterarbeit [✓]

Name und Vorname: Gomez Mendez Jose Gerardo

Matrikelnummer: 3097036

Studiengang/ Studienrichtung: Master in Mechatronics

Anschrift: Schillerstrasse 86 Zimmer 2319

Geburtsdatum und -ort: 20.08.1992 Xalapa, Mexiko

Telefon und Mailadresse: 017672828756 josegerardo@gomezmendez.com

Betreuende/r Prof.: Wollert

Korreferent (falls = externer Betreuer, s.u.):

Externer Betreuer (falls zutreffend; Titel, Name, Unternehmen bzw. Institution, Tel., Email):

Titel der Abschlussarbeit (max. 100 Zeichen inkl. Leerzeichen.Bitte in Normschrift oder maschinell ausfüllen; bitte beachten Sie: der Titel kann nach der Anmeldung nur auf Antrag innerhalb der ersten 4 Wochen geändert werden):
Design and Development of a 3D Dynamic Configuration Interface for an Industry 4.0 Assebly Cell

Ich erkläre durch meine Unterschrift, dass ich die Voraussetzungen für die Zulassung zur o.a. Abschlussarbeit erfülle und die geforderten Unterlagen dem Prüfungssekretariat des Fachbereichs Maschinenbau und Mechatronik der FH Aachen bereits vorliegen oder dem Antrag beigefügt sind.Ich erkläre durch meine Unterschrift, dass ich bisher noch keinen Versuch zur Ablegung einer Abschlussarbeit unternommen habe. Mir ist bekannt, dass mit dem Datum zur Unterschrift meines Betreuers die Frist zur Abgabe der Abschlussarbeit beginnt!

20.10.2017 [illegible]
Datum/Unterschrift der/des **Studierenden**

Ich erkläre mich bereit, die Abschlussarbeit zu betreuen.

20.10.2017 [illegible]
Datum/Unterschrift der/des **betreuenden Prof.**

Wird die Abschlussarbeit im **Ausland** durchgeführt? Ja [✓ Nein]

Falls ja: Datum/Unterschrift der/des **Auslandsbeauftragten**

Antrag eingegangen:

20.10.17 [illegible]
Datum/Unterschrift **Prüfungssekretariat**

83143 Mechatr. Projekt

Annexure 3

Class “ConfigureCar” Documentation

Class ConfigureCar
Script name: **configure_car.js** Asset id: **10213443** var ConfigureCar = pc.createScript('configureCar'); This script contains all the methods to change the configuration of the car. Add, modify or delete bricks.

Enum Attributes Detail

chassisEntity
Stores an instance of the class entity “Chassis”. Type: <**Chassis**> Visibility: **public** Default value: **null**

bricksEntity
Stores an instance of the class entity “Bricks”. Type: <**Bricks**> Visibility: **public** Default value: **null**

bricksListEntity
Stores an instance of the class entity “BricksList”. Type: <**BricksList**> Visibility: **public** Default value: **null**

availableColors
Stores an instance of the class “AvailableColors”.

Type: <**AvailableColors**>
Visibility: **public**
Default value: **null**

Coordinates

Stores an instance of the class “Coordinates”.

Type: <**Coordinates**>
Visibility: **public**
Default value: **null**

Enum Methods Detail

Initialize

ConfigureCar.prototype.initialize = function()
Initialize code called once per entity,

Update

ConfigureCar.prototype.update= function(dt)
Update code called every frame.

Parameters:

Dt	double	Variable used as a timer.

newBrick

ConfigureCar.prototype.newBrick = function(brickname,x,y,z,rotation_z,color,brick_type,ocvalue,code)
Adds a new brick to the car and set its respective parameters.

Parameters:

Brickname	double	The name of the brick. e.g. 2x2x2. See the documentation for a description of all the possible bricks.

X	Int	The x coordinate of the brick in grid coordinates. See documentation for a better understanding of the grid coordinates.
Y	Int	The y coordinate of the Brick in grid coordinates. See documentation for a better understanding of the grid coordinates.
Z	int	The z coordinate of the brick in grid coordinates. See documentation for a better understanding of the grid coordinates.
rotation_z	int	The rotation of the brick in the z-axis, given in an integer format (3). See documentation for a better understanding of the grid coordinates.
Color	String	The color of the brick. See the documentation for a list of all the available colors with their respective name.
brick_type	String	Defines the kind of brick. It can be fixed, customizable or optional depending on the possibility to edit it or delete it.
Ocvalue	int	The value of the radio button input in OpenCart. Every brick has a different value. See the documentation for the instruction on how to get this value.
Code	String	Unique identifier for a piece. It can be any string or a null value.

setChassisColor

ConfigureCar.prototype.setChassisColor = function(color, ocvalue, code)
Set the color of the chassis and defines its properties.

Parameters:

Color	String	The color of the chassis. See the documentation for a list of all the available colors with their respective name.
Ocvalue	int	The value of the radio button input in OpenCart. Every chassis has a different

		value. See the documentation for the instruction on how to get this value.
Code	String	Unique identifier for a piece. It can be any string or a null value.

resetChassisColor

ConfigureCar.prototype.resetChassisColor= function()
Changes the color of the chassis to the original color (the one that had before it was shown as selected)

setBrickColor

ConfigureCar.prototype.setBrickColor= function(brickInstance,color,ocvalue,code)
Sets the color of a brick and update the properties: color and code
@see newBrick()

Parameters:

brickInstance	Object	The entity containing the brick to be edited.
Color	String	The color of the brick.
Ocvalue	int	The value of the radio button input in OpenCart. Every brick has a different value.
Code	String	Unique identifier for a piece. It can be any string or a null value.

setBrickSelectedColor

ConfigureCar.prototype.setBrickSelectedColor= function(brickInstance)
Shows the brick in editable mode (the brick looks transparent), but it doesn't affect its properties.

Parameters:

brickInstance	Object	The entity containing the brick to show as selected.

setBrickPosition		
ConfigureCar.prototype.setBrickPosition = function(brickInstance,x,y,z,rotation_z) Define the position of the brick in terms of grid coordinates (integer number) See the coordinates documentation for the description of the grid. **Parameters**:		
Brickname	double	The entity containing the brick to be edited.
X	int	The x coordinate of the brick in grid coordinates.
Y	int	The y coordinate of the brick in grid Coordinates.
Z	int	The z coordinate of the brick in grid coordinates.
rotation_z	int	The rotation of the brick in the z axis, given in an integer format (3).

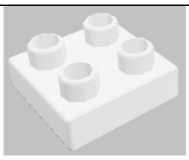
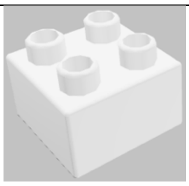
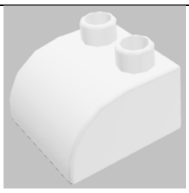
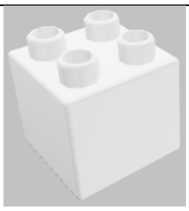
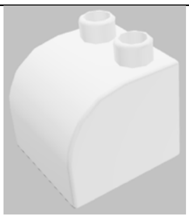
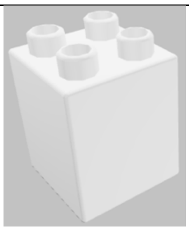
Annexure 4
Available Colors

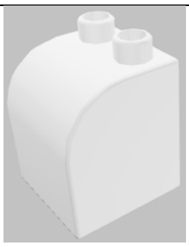
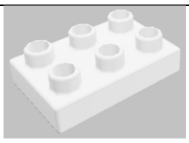
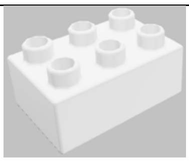
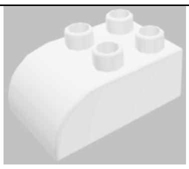
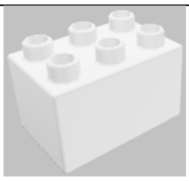
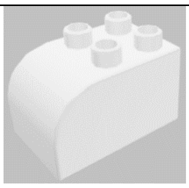
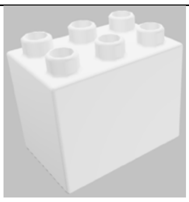
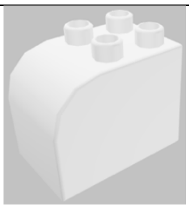
#	Color Name	Image	HTML Color Code	RGB		
1	Black	⬛	1B2A34	0	0	0
2	Blue	🟦	26469A	38	70	154
3	Brown	🟫	7B5D41	123	93	65
4	DarkBlue	🟦	19325A	25	50	90
5	DarkGreen	🟩	00852B	0	133	43
6	DarkGrey	⬛	545955	84	89	85
7	DarkOrange	🟧	91501C	145	80	28
8	DarkPink	🟥	D3359D	211	53	157
9	DarkPurple	🟪	441A91	68	26	145
10	DarkYellow	🟧	DD982E	221	152	46
11	Green	🟩	58AB41	88	171	65
12	Grey	⬜	8A928D	138	146	141
13	Lavender	🟪	CDA4DE	205	164	222
14	LightBlue	🟦	97CBD9	151	203	217
15	LightBrown	🟫	7B5D41	182	147	116
16	LightGreen	🟩	A5CA18	165	202	24
17	LightGrey	⬜	BCB4A5	188	180	165

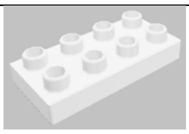
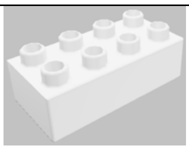
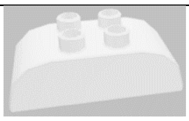
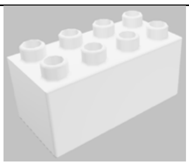
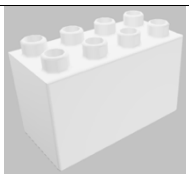
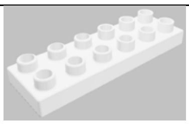
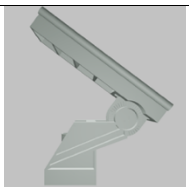
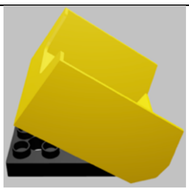
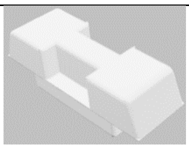
18	LightOrange	🟧	FCAC00	252	172	0
19	LightPurple	🟪	8A12A8	138	18	168
20	LightYellow	🟨	FFD67F	255	214	127
21	Magenta	🟪	901F76	144	31	118
22	MediumBlue	🟦	7396C8	115	150	200
23	Orange	🟧	D67923	214	121	35
24	Pink	🟪	FF9ECD	255	158	205
25	Purple	🟪	671F81	103	31	129
26	Red	🟥	B40000	180	0	0
27	Tan	🟫	B0A06F	176	160	111
28	Turquoise	🟦	069D9F	6	157	159
29	White		FFFFFF	255	255	255
30	Yellow	🟨	FAC80A	250	200	10

Table 1 Available Colors

Annexure 5
Available Brick Types

#	Image	Brick Name for Template Definition	Stud Dimensions		
			Width	Length	Height
1		2x2x1	2	2	1
2		2x2x2	2	2	2
3		2x2x2-bow	2	2	2
4		2x2x3	2	2	3
5		2x2x3-bow	2	2	3
6		2x2x4	2	2	4

7		2x2x4-bow	2	2	4
8		2x3x1	2	3	1
9		2x3x2	2	3	2
10		2x3x2-bow	2	3	2
11		2x3x3	2	3	3
12		2x3x3-bow	2	3	3
13		2x3x4	2	3	4
14		2x3x4-bow	2	3	4

15		2x4x1	2	4	1
16		2x4x2	2	4	2
17		2x4x2-bows	2	4	2
18		2x4x3	2	4	3
19		2x4x4	2	4	4
20		2x6x1	2	6	1
21		2x2x4-ladder	2	2	4
22		2x4x4-dumper	2	4	4
23		2x1x2-emergencylight	2	1	2

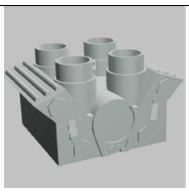
24		2x2x2-engine	2	2	2
25		2x4x4-petroltank	2	4	4
26		2x2x4-towtruck	2	4	4

Table 2 Available Bricks

Annexure 6

Example of the values for a Lego Duplo car

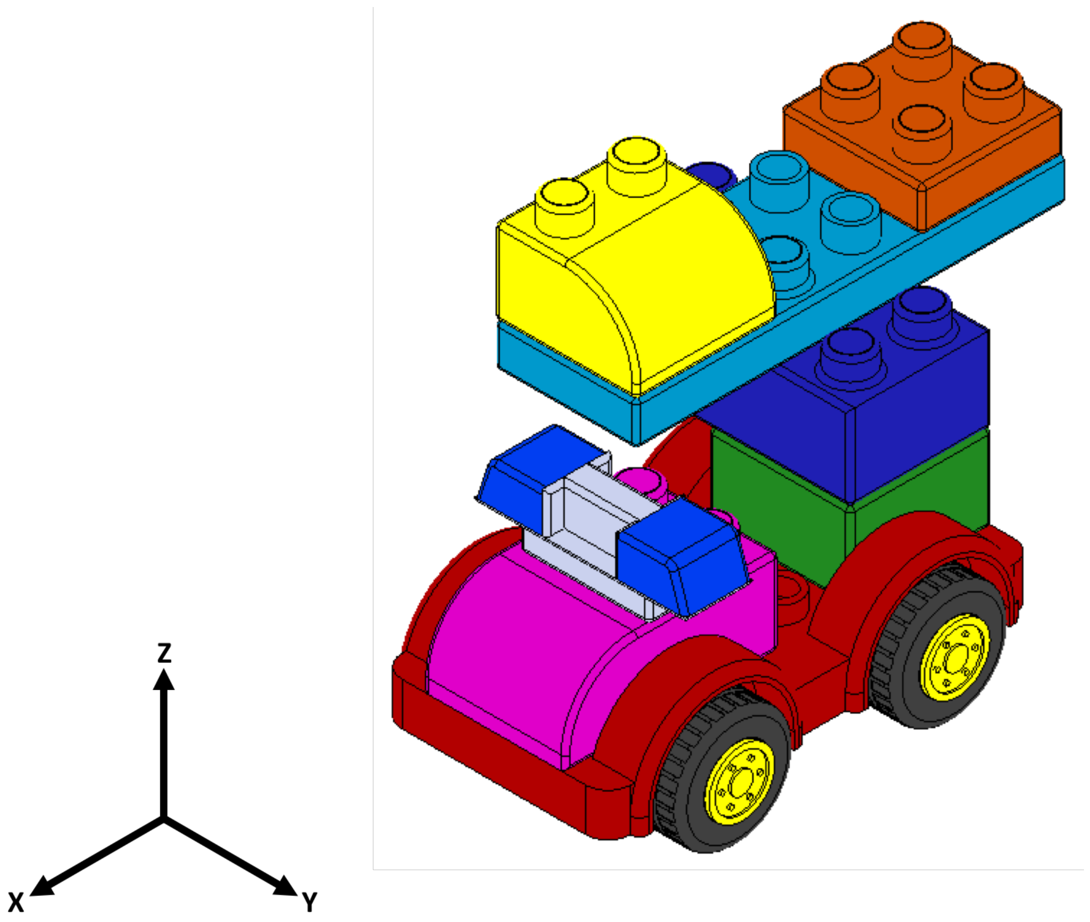


Figure 56 Lego Duplo Car example

Piece Name	Color	x_position	y_position	z_position	rotation_z
Chassis	red	0	0	0	0
2x3x2-bow	dark pink	3	0	1	0
2x2x2	dark green	-4	0	1	0
2x1x2-emergencylight	blue	3	0	3	0
2x4x2	dark blue	-4	-2	3	3
2x6x1	turquoise	-4	-2	5	0
2x2x2-bow	yellow	0	-2	6	1
2x2x1	orange	-8	-2	6	0

Table 3 Lego Duplo Car values for Figure 56

Annexure 7
How to get the ocvalue in OpenCart

In order to get the names and values for each piece from OpenCart©, is necessary to use the “Developer tools” of a web browser. In this case, Google Chrome was used. It is important to mention that this has to be done either before the plugin was installed or when the modification is disabled.

The first step is to enable the “Inspect Element mode” as shown in Figure 57.

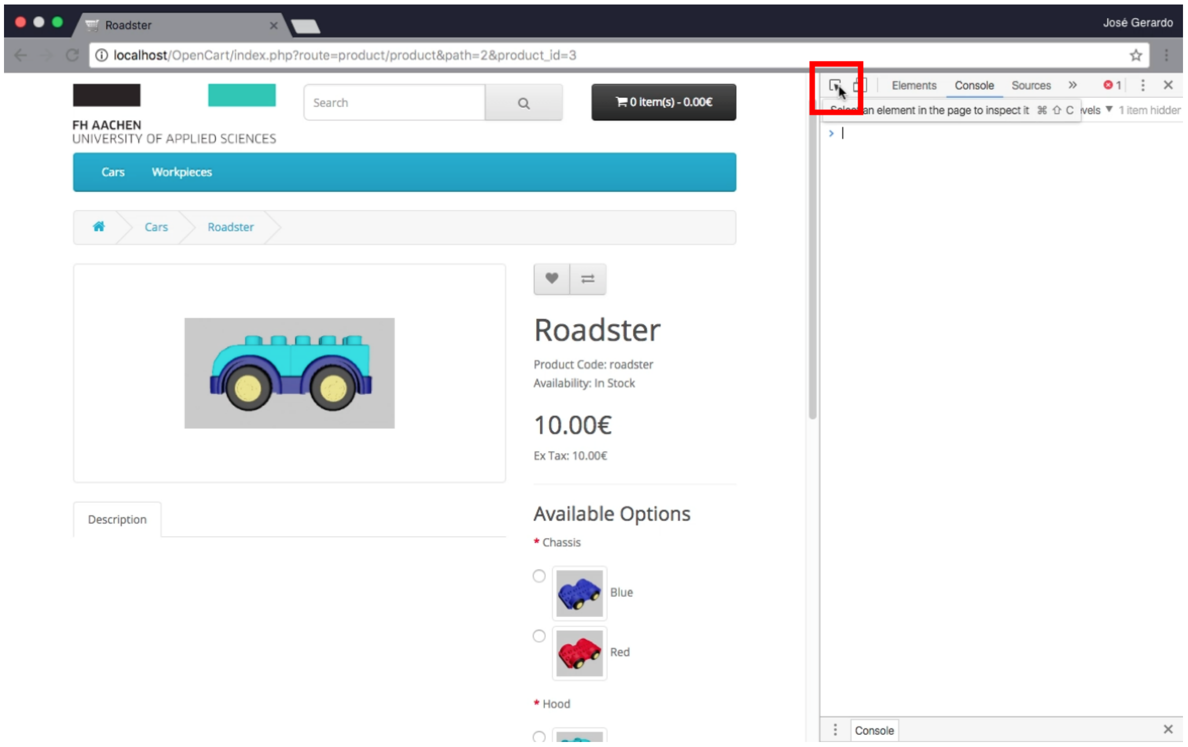


Figure 57 Step 1

The second step is to click the radio button of the desired element, like in Figure 58. This shows all the elements of the web page, and it marks the lines of code that references the object as in Figure 59; accordingly to this example, the obtained value is “13”. This number is related to the blue chassis in the roadster model. The last step is to go iteratively to each element on the car.

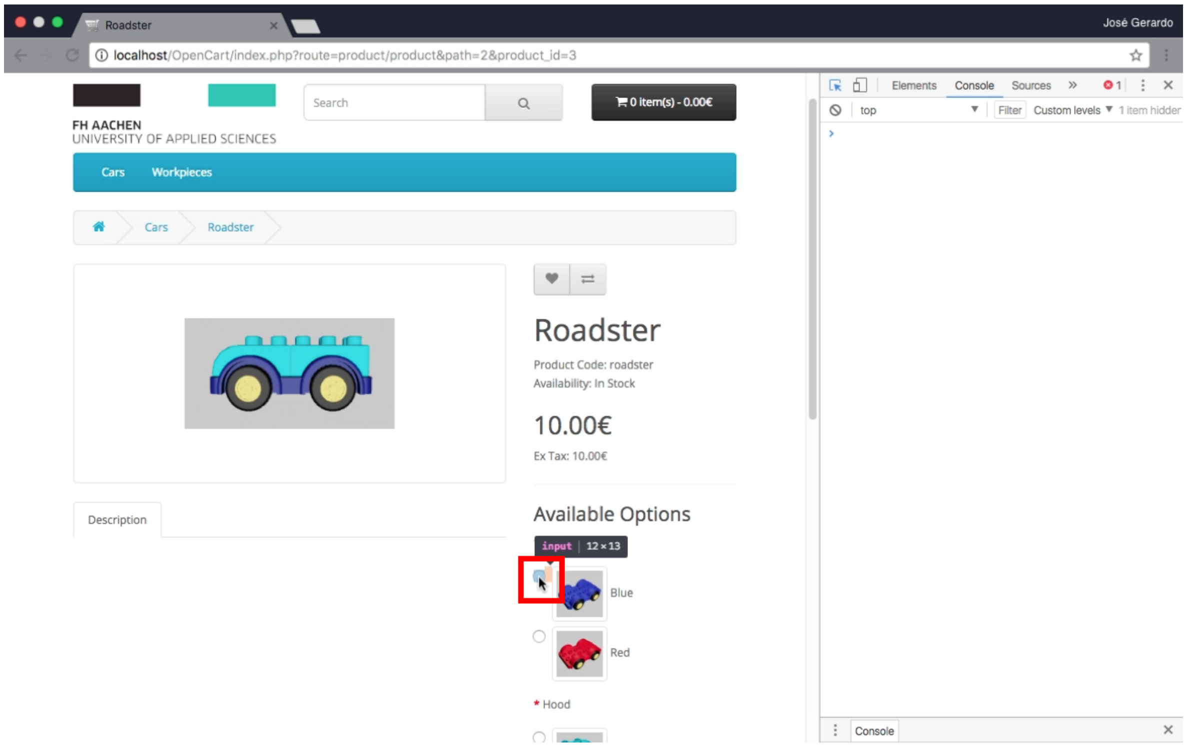
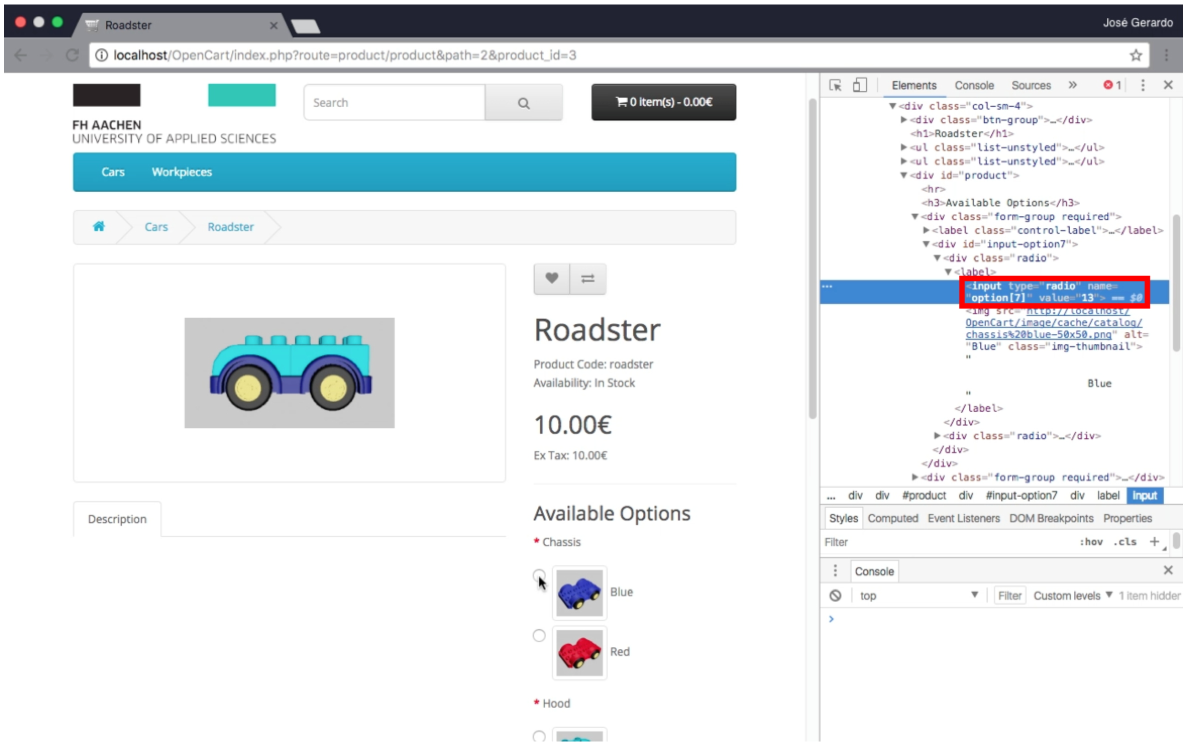


Figure 58 Step 2



```
▼<div class="col-sm-4">
  ▶<div class="btn-group">…</div>
   <h1>Roadster</h1>
  ▶<ul class="list-unstyled">…</ul>
  ▶<ul class="list-unstyled">…</ul>
  ▼<div id="product">
     <hr>
     <h3>Available Options</h3>
    ▼<div class="form-group required">
      ▶<label class="control-label">…</label>
      ▼<div id="input-option7">
        ▼<div class="radio">
          ▼<label>
             <input type="radio" name=
             "option[7]" value="13"> == $0
             <img src="http://localhost/
             OpenCart/image/cache/catalog/
             chassis%20blue-50x50.png" alt=
             "Blue" class="img-thumbnail">
             "

                         Blue
             "
           </label>
         </div>
        ▶<div class="radio">…</div>
       </div>
     </div>
    ▶<div class="form-group required">…</div>
```

… div div #product div #input-option7 div label input

Figure 59 Step 3

Annexure 8

Contents of the DVD

In the DVD at the end of this document, the following files are included:

1. A “pdf” version of this thesis.
2. The file “3DViewer.ocmod.zip” that contains:
   2.1. The OpenCart modification including the Viewer as a plugin
   2.2. The “README.txt” file with the installing instructions.
   2.3. The “Configuration Manual for Implementing the 3D Viewer as a Plugin for OpenCart”.
3. The folder “3DViewer” that includes:
   3.1. The full-page version of the Viewer.
   3.2. The classes documentation in an HTML format in the folder “ClassesDocumentation”.
   3.3. All the diagrams presented in an HTML format inside the folder “Diagrams”.
   3.4. The “Configuration Manual for Using the 3D Viewer outside OpenCart”.
4. A virtual machine running the full implementation of the Industry 4.0 system, including the plugin installed in OpenCart.
5. The folder “CAD bricks” containing all the available bricks 3D models in “obj” and “fbx” format.