

Univerza v Ljubljani  
Fakulteta *za gradbeništvo*  
*in geodezijo*



**GABRIEL PEREIRA DE CARVALHO VELOSO**

**WEB3 SERVICES FOR INFORMATION EXCHANGE IN THE  
CONSTRUCTION SITE**

**STORITVE WEB3 ZA IZMENJAVO INFORMACIJ  
NA GRADBIŠČU**



European Master in  
Building Information Modelling

Master thesis No.:

Supervisor:  
Prof. Boštjan Brank, Ph.D.

Co-supervisor:  
Prof. Vlado Stankovski, Ph.D.

Ljubljana, 2023



Co-funded by the  
Erasmus+ Programme  
of the European Union

## **ERRATA**

| <b>Page</b> | <b>Line</b> | <b>Error</b> | <b>Correction</b> |
|-------------|-------------|--------------|-------------------|
|-------------|-------------|--------------|-------------------|

*»This page is intentionally blank«*

## **BIBLIOGRAFSKO – DOKUMENTACIJSKA STRAN IN IZVLEČEK**

**UDK:** 004.774:69.055(043.3)

**Avtor:** Gabriel Pereira de Carvalho Veloso

**Mentor:** Prof. dr. Boštjan Brank

**Somentor:** Prof. dr. Vlado Stankovski

**Naslov:** Storitve Web3 za izmenjavo informacij na gradbišču

**Tip dokumenta:** magistrsko delo

**Obseg in oprema:** 75 str., 79 sl., 7 pregl.

**Ključne besede:** BIM, IDS, Samostojna suverena identiteta, DID, NFT, Blockchain, ISO19650

**Izvleček:** V industriji AEC posledice neučinkovite izmenjave informacij odmevajo med projekti, vplivajo na časovne načrte, proračune in splošno kakovost, zato so potrebne rešitve za opredelitev predvidljivih in zanesljivih procesov dostave podatkov. Ta raziskava ocenjuje uporabo storitev Web3 v kombinaciji z delovnimi tokovi Open BIM za izmenjavo informacij v kontekstu pametnih konstrukcij. Ključna referenca za to delo je evropski projekt BUILDCHAIN, ki si prizadeva ustvariti vrhunsko tržnico, ki omogoča akterjem v sektorju AEC, da delijo svoje ponudbe, certifikate kakovosti in poverilnice. Na ta način ta raziskava raziskuje napredne funkcije, kot je integracija med zbirkami NFT (nezamenljivih žetonov), DID (decentralizirani identifikatorji) in IDS (specifikacija dostave informacij), ki pridobivajo podatke iz modelov IFC. Izvedba teh integracij lahko nudi mehanizme za podporo vzpostavitvi informacijske potrebe, za zagotavljanje ustrezne količine podatkov ustreznim akterjem in za potrjevanje dostave, kar zagotavlja avtomatizacijo in zaupanje v gradbene procese. Študija bo potekala v štirih metodoloških korakih: (1) naj sodobnejša analiza s ciljem zbrati zahteve in strukturirati osnove za ustvarjanje konceptualne zasnove za decentraliziran trg gradbenih storitev; (2) podrobna zasnova rešitve za določen primer uporabe, ki poudarja podatkovni model aplikacije in tehnološko široko sliko; (3) implementacija in predstavitev zaledja in sprednjega dela decentralizirane aplikacije z uporabo Reacta, Internet Computer Blockchain in IFC.js; (4) ocena uporabnosti v gradbeni praksi in dodana vrednost BUILDCHAIN. Rezultat preiskave je spletna aplikacija za tržnico NFT, kjer lahko naročniki dodelijo storitve izvajalcem, pri čemer določijo pregledne zahteve, storitvam pa je mogoče slediti prek sistemov ugleda. Zato so rezultati, načrtovani s tem razvojem, zagotoviti večjo preglednost postopka razpisa z decentralizacijo informacij, pravilno slediti dejavnostim, povezanim z življenjskimi cikli zgradbe, in vzpostaviti delovne tokove za definiranje informacijskih zahtev in skladnost s standardom ISO19650 za primere uporabe BUILDCHAIN, ki temeljijo na decentraliziranih tehnologijah.

*»This page is intentionally blank«*

## **BIBLIOGRAPHIC– DOKUMENTALISTIC INFORMATION AND ABSTRACT**

**UDC:** 004.774:69.055(043.3)

**Author:** Gabriel Pereira de Carvalho Veloso

**Supervisor:** Prof. dr. Boštjan Brank

**Co-supervisor:** Prof. dr. Vlado Stankovski

**Title:** Web3 Services for Information Exchange in the Construction Site.

**Document type:** Master Thesis

**Scope and tools:** 75 p., 79 fig., 7 tab.

**Keywords:** BIM, IDS, Self-Sovereign Identity, DID, NFT, Blockchain, ISO19650

**Abstract:** In the AEC Industry, the repercussions of inefficient information exchange reverberate across projects, impacting timelines, budgets, and overall quality, thus, solutions to define predictable and reliable data delivery processes are needed. This research evaluates the use of Web3 services combined with Open BIM workflows for information exchange in the context of Smart Constructions. A key reference for this work is the European Project BUILDCHAIN, which aspires to create a cutting-edge marketplace that allows AEC sector actors to share their offerings, quality certificates, and credentials. In this way, this research explores advanced features such as the integration between NFT (non-fungible token) collections, DIDs (Decentralized Identifiers), and IDS (Information Delivery Specification) obtaining data from IFC Models. The execution of these integrations can offer mechanisms to support the establishment of information needs, supply the appropriate amount of data to the appropriate actors, and validate the delivery, ensuring automation and trust in the construction processes. The study will proceed in four methodological steps: (1) state-of-the-art analysis with the objective of gathering requirements and structuring the bases to generate a conceptual design for a decentralized marketplace for construction services; (2) detailed design of the solution for the specific use case, highlighting the data model of the application and the technological big picture; (3) implementation and demonstration of the Back end and Front end of the Decentralized App using React, Internet Computer Blockchain and IFC.js; (4) usability assessment in construction practice and value added to the BUILDCHAIN. The outcome of the investigation is a web application for an NFT Marketplace where Clients can assign services to Contractors, establishing transparent requirements, and the services can be tracked through reputation systems. Therefore, the results aimed with this development are to provide more transparency for Tendering process through the decentralization of information, properly track activities linked to building life cycles, and, establish workflows for defining information requirements and ISO19650 compliance to the BUILDCHAIN use cases relying on decentralized technologies.

*»This page is intentionally blank«*

## **ACKNOWLEDGEMENTS**

First, I would like to thank all my family, my parents José Henrique and Jaqueline, and my brother Rafael, not only for supporting me on the idea of coming to this Master's but for all the experiences that they provided to me along my life which helped me to get here.

Then, all the new friends that this program brought, incredible people, sharing common objectives and wills. In special, Mo, Fabri, Paula, Joe, Cristi, Minja, Cesar, Vicente, a group that somehow made me feel at home in the hardest moments. Also, to Professor Vlado for the guidance and professors Tomo and Miguel for organizing this program.

I could not forget to mention my coworkers and friends at MRV, who helped a lot throughout this year by showing up for me when I needed them. A special thanks to Augusto, Isa, Ana, and Bárbara. This was the most intense year of my life until now, being away from Brazil, working from different time zones, and putting all I had into this Master, without all of you definitely I could not have made it.



*»This page is intentionally blank«*

## TABLE OF CONTENTS

|  |            |
|--|------------|
| <b>ERRATA</b> .....  | <b>I</b>   |
| <b>BIBLIOGRAFSKO – DOKUMENTACIJSKA STRAN IN IZVLEČEK</b> .....       | <b>III</b> |
| <b>BIBLIOGRAPHIC– DOKUMENTALISTIC INFORMATION AND ABSTRACT</b> ..... | <b>V</b>   |
| <b>ACKNOWLEDGEMENTS</b> .....  | <b>VII</b> |
| <b>TABLE OF CONTENTS</b> .....                                       | <b>IX</b>  |
| <b>INDEX OF FIGURES</b> .....  | <b>XI</b>  |
| <b>1 INTRODUCTION</b> .....  | <b>1</b>   |
| 1.1 Research and Objectives .....                                    | 2          |
| 1.2 Structure of the Document .....                                  | 2          |
| <b>2 RELATED WORKS</b> .....   | <b>3</b>   |
| 2.1 ISO 19650 SERIES .....   | 3          |
| 2.1.1 Project Information Model and Information Requirements .....   | 4          |
| 2.1.2 Level of Information Need.....                                 | 7          |
| 2.2 Open BIM Workflows.....  | 8          |
| 2.2.1 IFC – Industry Foundation Classes .....                        | 9          |
| 2.2.2 IDS – Information Delivery Specification .....                 | 12         |
| 2.3 BIM Data transfer methods and parsing.....                       | 16         |
| 2.4 Web3 Technologies in the AEC Industry.....                       | 19         |
| 2.4.1 Blockchain and Decentralized Applications .....                | 21         |
| 2.4.2 Self-Sovereign Identities .....                                | 22         |
| 2.4.3 NFT – Non-Fungible Tokens .....                                | 25         |
| 2.4.4 Decentralized Knowledge Graphs.....                            | 26         |
| <b>3 METHODOLOGY</b> .....   | <b>27</b>  |
| 3.1 Requirements Gathering and Analysis.....                         | 27         |
| 3.2 Use Case Development for the Decentralized Marketplace.....      | 28         |
| 3.3 Detailed Design of the Solution .....                            | 30         |

|          |   |           |
|----------|---|-----------|
| 3.3.1    | Data Model .....  | 32        |
| <b>4</b> | <b>IMPLEMENTATION AND DEMONSTRATION .....</b>   | <b>34</b> |
| 4.1      | Technology Stack and Tools .....  | 34        |
| 4.2      | Integration of IFC.js viewer and IDS (Information Delivery Specification).....                    | 35        |
| 4.3      | Integration with Blockchain, Smart Contracts, and NFT Collections .....                           | 44        |
| 4.4      | Frontend of the Decentralized Marketplace .....   | 52        |
| 4.5      | Integration of DID Documents .....  | 53        |
| 4.6      | Using the Application Step by Step.....   | 56        |
| 4.6.1    | Assessment and Need .....   | 56        |
| 4.6.2    | Invitation to Tender .....  | 57        |
| 4.6.3    | Tender Response.....  | 59        |
| 4.6.4    | Appointment.....  | 60        |
| 4.6.5    | Mobilization .....  | 61        |
| 4.6.6    | Collaborative Production of Information.....  | 62        |
| 4.6.7    | Information Model Delivery.....   | 64        |
| 4.6.8    | Project Close-out .....   | 64        |
| <b>5</b> | <b>RESULTS AND DISCUSSION.....</b>  | <b>65</b> |
| 5.1      | Usability Assessment.....   | 65        |
| 5.2      | Discussion on the Effectiveness .....   | 67        |
| 5.3      | Added Values to the BUILDCHAIN Project .....  | 68        |
| <b>6</b> | <b>CONCLUSION AND FUTURE WORKS.....</b>   | <b>69</b> |
| 6.1      | Limitations and Future Works .....  | 69        |
| 6.1.1    | Integration with ONTOCHAIN framework .....  | 70        |
| 6.1.2    | IoT, Edge-computing, and Containerization .....   | 70        |
| 6.1.3    | Integration with buildingSMART Data Dictionary (bSDD) and BIM Collaboration<br>Format (BCF) ..... | 71        |
| 6.2      | Contributions and Final Thoughts .....  | 71        |
|          | <b>REFERENCES .....</b>   | <b>73</b> |

## INDEX OF FIGURES

|   |    |
|---|----|
| Figure 1: Actors by the ISO19650 (UK BIM Framework, 2022a) .....  | 4  |
| Figure 2: Project Phases by the ISO19650 (UK BIM Framework, 2022a).....   | 5  |
| Figure 3: Information Requirements (UK BIM Framework, 2022b).....   | 5  |
| Figure 4: Exchange Information Requirements and Information Delivery (UK BIM Framework, 2022b)<br>.....                     | 7  |
| Figure 5: Information Requirements and Delivery (UK BIM Framework, 2022b) .....   | 8  |
| Figure 6: IFC 4.0.2.1 Data Schema Architecture (buildingSMART International, 2017).....                                     | 10 |
| Figure 7: IFC EXPRESS to ifcOWL (buildingSMART International, 2016).....  | 11 |
| Figure 8: Printout of the RDF graph representation for the <i>IfcPropertySetDefinitionSet</i> (Pauwels & Terkaj, 2016)..... | 11 |
| Figure 9: Brief representation of the overall ifcJSON4 schema structure(Afsari et al., 2017). .....                         | 12 |
| Figure 10: The expansion of MVDs and EIRs adding data from outside an MVD (Berlo, 2019).....                                | 13 |
| Figure 11: IDS macro flow (buildingSMART International, 2020).....  | 13 |
| Figure 12: Snapshot of an IDS file.....   | 15 |
| Figure 13: Comparison between each data transfer method (Pauwels et al., 2023).....   | 16 |
| Figure 14: LBDviz system architecture (Donkers A. and Yang, 2023) .....   | 19 |
| Figure 15: The primary roles involved with the exchange of verifiable credentials (Preukschat et al., 2021) .....           | 23 |
| Figure 16: Relationships between the DID, DID document, and DID subject (Preukschat et al., 2021)<br>.....                  | 24 |
| Figure 17: Class Diagram of the system (Cocco et al., 2022).....  | 25 |
| Figure 18: Business Process Model .....   | 28 |
| Figure 19: Zoom into the Stages 5.1 and 5.2.....  | 28 |
| Figure 20: Zoom into the Stages 5.3 and 5.4.....  | 29 |
| Figure 21: Zoom into the Stages 5.5 and 5.6.....  | 30 |
| Figure 22: Zoom into the Stages 5.7 and 5.8.....  | 30 |
| Figure 23: System Architecture.....   | 31 |
| Figure 24: High-Level Ontology.....   | 33 |
| Figure 25: Generation of the IFC.js viewer.....   | 35 |
| Figure 26: Loading the IFC Model .....  | 35 |
| Figure 27: Specification Object.....  | 36 |
| Figure 28: Specification Definition Object .....  | 36 |
| Figure 29: Facet Object.....  | 36 |
| Figure 30: Facet Parameter Object.....  | 37 |
| Figure 31: Method to parse the Specification Data.....  | 37 |

|   |    |
|---|----|
| Figure 32: Method to Obtain the Facet and Parameters of the Applicability or Requirement .....        | 38 |
| Figure 33: Function to Query the Elements by the Entity Facet .....                                   | 39 |
| Figure 34: Method to Query the Elements by the Attributes Facet .....                                 | 39 |
| Figure 35: Method to Query the Elements by the Materials .....  | 40 |
| Figure 36: Method to Query the Elements by the Properties Facet.....                                  | 41 |
| Figure 37: Method to Query the Elements by the Classification Facet.....                              | 42 |
| Figure 38: Method to Query the Elements by the PartOf Facet .....                                     | 42 |
| Figure 39: Switch Case Statement to Select the Query Methods Based on the IDS parsed.....             | 43 |
| Figure 40: Visualization of the Implementation.....   | 44 |
| Figure 41: Console logs of the test .....   | 44 |
| Figure 42: NFT Canister Candid File .....   | 45 |
| Figure 43: <i>OpenBIM_D</i> Canister Candid File.....   | 46 |
| Figure 44: Query Function Example .....   | 46 |
| Figure 45: Function Mint.....   | 47 |
| Figure 46: Creating Actors .....  | 47 |
| Figure 47: Integration between NFT Metadata and IFC Entities .....                                    | 48 |
| Figure 48: <i>IfcOrganization</i> EXPRESS Specification (buildingSMART International, 2019).....      | 48 |
| Figure 49: Function to create <i>IfcOrganization</i> Entity.....                                      | 49 |
| Figure 50: <i>IfcActorRole</i> EXPRESS Specification (buildingSMART International, 2019).....         | 49 |
| Figure 51: Function to create <i>IfcActorRole</i> Entity.....   | 49 |
| Figure 52: Console log of <i>IfcOrganization</i> with Role.....                                       | 50 |
| Figure 53: <i>IfcActor</i> EXPRESS Specification (buildingSMART International, 2019) .....            | 50 |
| Figure 54: Function to Create <i>IfcActor</i> entity (buildingSMART International, 2019).....         | 50 |
| Figure 55: Console log of <i>IfcActor</i> Entity (buildingSMART International, 2019) .....            | 51 |
| Figure 56: <i>IfcRelAssignsToActor</i> EXPRESS Specification (buildingSMART International, 2019)..... | 51 |
| Figure 57: Function to create <i>IfcRelAssignsToActor</i> Entity .....                                | 51 |
| Figure 58: React Components Diagram .....   | 52 |
| Figure 59: Integration of DID Documents .....   | 53 |
| Figure 60: <i>IfcApproval</i> EXPRESS Specification (buildingSMART International, 2019) .....         | 54 |
| Figure 61: Create <i>IfcApproval</i> Entity Function .....  | 54 |
| Figure 62: Console log of the <i>IfcApproval</i> .....  | 55 |
| Figure 63: Create <i>IfcRelAssociatesApproval</i> Entity function.....                                | 55 |
| Figure 64: First Floor of the Building .....  | 56 |
| Figure 65: IDS File for Concrete Walls.....   | 57 |
| Figure 66: Minter page .....  | 58 |
| Figure 67: Notification of the Minted NFT .....   | 58 |
| Figure 68: Sending NFTs to Tender.....  | 59 |

|  |    |
|--|----|
| Figure 69: NFT in tender as presented for the Owner and in the Market .....  | 60 |
| Figure 70: Confirm Lead Appointed Party Button.....                          | 61 |
| Figure 71: My Models Webpage Lead Appointed Party View.....                  | 62 |
| Figure 72: Updating properties. ....   | 63 |
| Figure 73: Request Approval Button .....                                     | 63 |
| Figure 74: Connecting the DID URL.....                                       | 63 |
| Figure 75: My Models webpage with the Appointing Party View. ....            | 64 |
| Figure 76: Giving Reputation to the Service.....                             | 64 |
| Figure 77: Design possibility for a IDS database.....                        | 66 |
| Figure 78: Reputation System Workflow.....                                   | 67 |
| Figure 79: High Level Ontology to Specify BUILDCHAIN Use Cases Metadata..... | 68 |

**INDEX OF TABLES**

|  |    |
|--|----|
| <b>Table 1:</b> List of Facets and its Parameters (Adapted from buildingSMART International, 2021) ..... | 15 |
| <b>Table 2:</b> Examples of Methods from the Web-ifc API .....   | 17 |
| <b>Table 3:</b> Examples of Methods from the Web-ifc-three API .....                                     | 18 |
| <b>Table 4:</b> Examples of Methods from the Web-ifc-viewer API .....                                    | 18 |
| <b>Table 5:</b> Atomic packs of development .....  | 32 |
| <b>Table 6:</b> React Components Interactions with Canisters .....                                       | 53 |
| <b>Table 7:</b> Limitations of the IDS Parser .....  | 69 |

## **LIST OF ACRONYMS**

AEC - Architecture, Engineering, and Construction

API – Application Programming Interface

BCF – BIM Collaboration Format

BIM – Building Information Modelling

bSDD – buildingSMART Data Dictionary

CDE – Common Data Environment

DApp – Decentralized Application

DID – Decentralized Identifiers

IDS – Information Delivery Specification

IFC – Industry Foundation Classes

IoT – Internet of Things

NFT – Non-Fungible Token

SSI – Self-Sovereign Identity

VC – Verifiable Credentials



*»This page is intentionally blank«*

## 1 INTRODUCTION

Digital technologies and cutting-edge concepts are transforming how buildings are designed, constructed, and managed in the Architecture, Engineering, and Construction (AEC) industry. Building Information Modeling (BIM), which uses information-rich 3D models to streamline collaboration and improve decision-making across the building life cycle, is a crucial enabler of this shift.

At the same time, blockchain technology and associated Web3 applications have arisen as a disruptive force, introducing decentralized, secure, and transparent platforms across a variety of industries. In this perspective, the BUILDCHAIN project, a ground-breaking initiative sponsored by Horizon Europe, is a major reference. It aspires to create a cutting-edge marketplace that allows AEC sector actors to share their offerings, quality certificates, and credentials while properly tracking all activities linked to building life cycles.

The cornerstone of the BUILDCHAIN vision is the Digital Building Logbook (DBL), a comprehensive repository designed to empower municipalities in governing and managing their building inventory, adopting a Decentralized Knowledge Graph (DKG), in pursuit of seamless data integration and novel functionalities.

This master thesis embarks on a research journey that draws inspiration from the innovative advancements prospected by BUILDCHAIN. The objective is to explore the integration of BIM and Web3 services within the AEC industry, with a specific focus on its added value and potential contributions to improve information exchange processes.

This research will evaluate the potential synergies of using Web3 services combined with Open BIM workflows for information exchange in the context of Smart Construction. It will aim to explore the integration of advanced features, such as Decentralized Identifiers (DIDs), Non-Fungible Token (NFT) collections, and Information Delivery Specification (IDS) in obtaining data from IFC models. With the macro-objective to ensure trust and automation in construction processes.

The expected outcome of this investigation is a web application for an NFT Marketplace, enabling transparent assignment of services, tracking of progress, and establishment of reputation systems. By enhancing information exchange, fostering collaboration, and improving efficiency, this application aims to revolutionize the construction industry by leveraging the benefits of decentralized technologies.

Overall, this research contributes to advancing the field of Open BIM workflows by harnessing the potential of Web3 services for more transparent, efficient, and trustworthy information exchange in construction processes, considering, as a reference, BUILDCHAIN objectives and use cases that will be further exposed in section 2.

## 1.1 Research and Objectives

The research follows four main steps. Firstly, a state-of-the-art analysis is conducted to understand the existing literature and practices in Web3 services, Open BIM workflows, and their application in Smart Construction. Requirements gathering and analysis are carried out to identify the specific needs and challenges in the industry, correlated with the BUILDCHAIN objectives.

Secondly, a conceptual design is developed, considering the integration of DIDs, NFT collections, and IDS with data derived from IFC models. This concept includes the development of specific use cases for a decentralized Marketplace for construction services, where Clients can allocate services to Contractors while setting clear specifications and information exchange protocols on the job site.

In the third step, a detailed design of the solution is developed. This includes the technical architecture, protocols, and standards for data exchange, ensuring interoperability and security. Then, the implementation and demonstration of the designed solution highlight the workflows of connecting web technologies and frameworks compatible with Web3 services to openBIM standards in compliance with ISO19650, through the decentralized marketplace.

The fourth step involves a usability assessment of the developed solution in real-world construction practices. This will proceed using real-world BIM Models, and information requirements from use cases defined in the BUILDCHAIN scope, comparing the implementation with other tools in the market in terms of effectiveness, cost, and simplicity to be used.

Through the decentralization of information, this development aims to increase transparency for the tendering process, accurately track building life cycle activities, and establish workflows for defining information requirements and ISO19650 compliance for BUILDCHAIN use cases relying on decentralized technologies.

## 1.2 Structure of the Document

This thesis is structured in five following sections. The next one, **section 2**, presents several related works to Information Management workflows related to the ISO 19650 and OpenBIM stack tools, integrations between BIM, blockchain, smart contracts, knowledge graphs, Self-sovereign Identities, and NFT technologies, that can leverage the implementation of the proposed Decentralized Marketplace. **Section 3** describes the methodology used, the user stories of the application, and its data model and architecture. **Section 4** outputs the technological approach used in every step of the development, and an example of use in the application, to present the step-by-step of the user. **Section 5** provides the usability assessment based on the example that was run in the platform, and the evaluation of the effectiveness. In the end, **section 6**, provides the conclusion of the work, analyzing limitations, future works, and other technologies that could be integrated.

## 2 RELATED WORKS

This review is broken down into three main topics that are the foundation of this research, which are Information Management workflows, represented by the ISO 19650 Series; OpenBIM approach and standards; BIM data transfer and parsing methodologies; and the use of Web3 technologies in the AEC Industry.

### 2.1 ISO 19650 SERIES

The ISO 19650 Series deals with the organization and digitization of information about buildings and civil engineering works, considering building Information management through building information modeling, which is key for the correct definition of approaches in this thesis. This standard is divided into five parts:

- **Part 1: Concepts and principles (2018)** - The document's goal is to provide a framework for managing information for all actors across the whole life cycle of any produced asset, including exchanging, documenting, versioning, and organizing.
- **Part 2: Delivery phase of the assets (2018)** - defines the need for information management in the form of a management procedure during the asset delivery phase and information exchanges throughout the process.
- **Part 3: Operational phase of the assets (2020)** - describes the requirements for information management in the form of a management procedure during the asset operational phase and information exchanges throughout the process.
- **Part 4: Information exchange (2022)** – to guarantee the quality of the resulting project information model or asset information model, describe the specific procedure and criteria for decision-making while performing an information exchange as described by the ISO 19650 standard. It describes how the ideas in ISO 19650-1 are put into practice and is applicable to any information exchange throughout the delivery stages covered by ISO 19650-2 and operational trigger events described by ISO 19650-3.
- **Part 5: Security-minded approach to information management (2020)** – discusses the procedures necessary to develop a security mentality and culture that is acceptable and proportionate across companies that have access to sensitive information, including the requirement to monitor and audit compliance.

For this review, to mainly understand the concept of the Level of Information Need declared in part 1, and the requirements for information governance along the delivery phase of assets, resulting in the Project Information Model, included in part 2.

### 2.1.1 Project Information Model and Information Requirements

The ISO 19650 Series defined specific terms to refer to roles and team contexts. The definition of roles is divided into three parties:

- **Appointing Party**, whoever needs the information—either the client or the person or organization handling it on their behalf.
- **Lead Appointed Party**, a party chosen by the client.
- **Appointed Party**, a party chosen by the lead appointed party, which provides the information.

This definition can be applied to different scenarios, such as multiple lead appointed parties and appointed parties or single-party relationships as demonstrated in Figure 1. The definition of team contexts is also divided into three:

- **Project team**, which involves all three parties.
- **Delivery team**, the combination of Lead Appointed party and Appointed parties.
- **Task team**, which are the Appointed parties only.

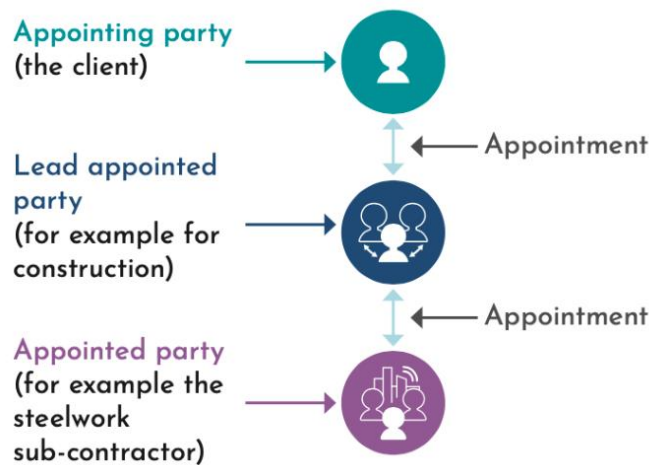


Figure 1: Actors by the ISO19650 (UK BIM Framework, 2022a)

The ISO 19650-2 in the fifth clause defines the project activities that each party is responsible for during the delivery phase. These activities are divided into eight stages which are shown in Figure 2. Each stage and its related activities can be categorized as “Project-level” (per project) or “Appointment-level” (per appointment). Stages 5.1 – Assessment and need, and 5.8 – Project close-out are the ones categorized as Project-level. Each party will have a higher focus in specific stages, which is distributed in this way:

- **Appointing Party**: Assessment and need, Invitation to tender, Project close-out.
- **Lead Appointed Party**: Tender response, Appointment, Mobilization, Collaborative production of information, Information model delivery.

- **Appointed Party:** Collaborative production of information, Information model delivery.

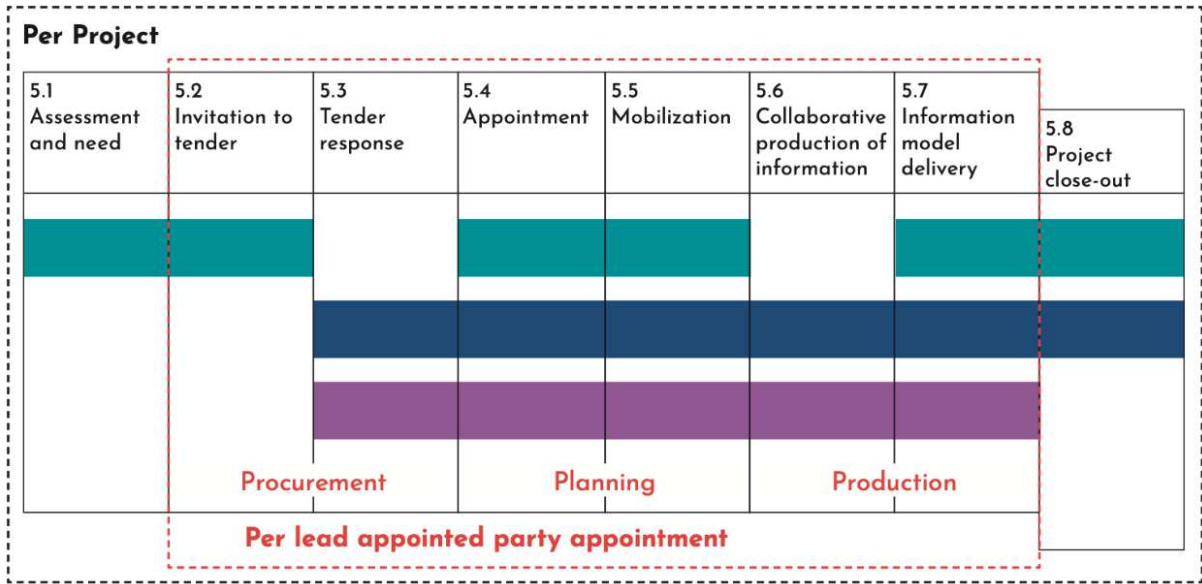


Figure 2: Project Phases by the ISO19650 (UK BIM Framework, 2022a)

Both ISO 19650 Parts 2 and 3 take into account information requirement structures that go from appointing to lead appointed parties and then to appointed parties, which also can be defined in the inverse direction and between delivery Teams. As illustrated in Figure 3, these two parts of the ISO define four structures of requirements that are correlated to each other, which specify the Information Models.

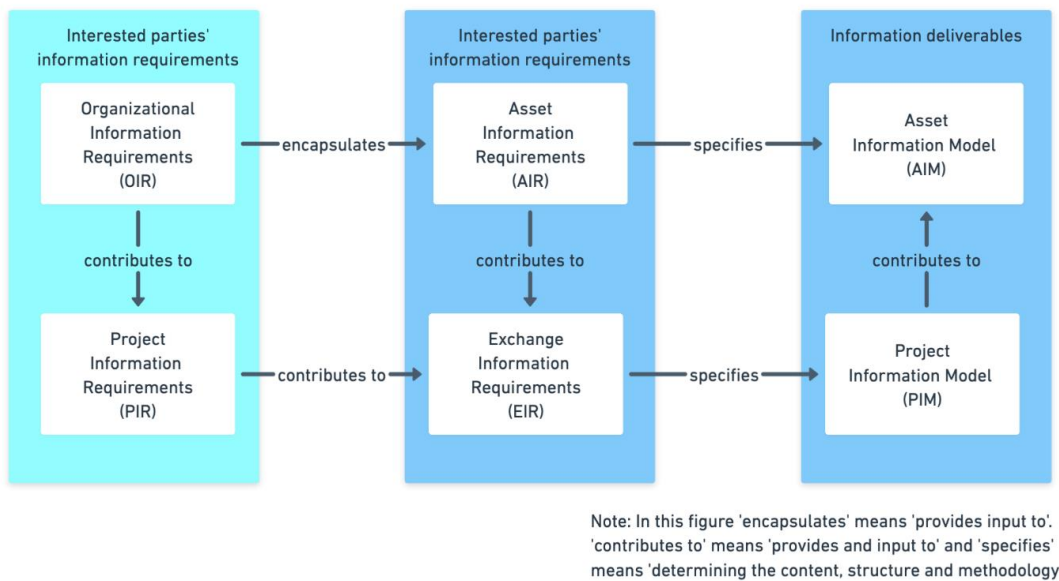


Figure 3: Information Requirements (UK BIM Framework, 2022b)

Each of these structures/documents has specific moments to be defined, responsible, objectives, components, and different strategic levels.

**Organizational Information Requirements** (UK BIM Framework, 2022b):

- It is seen as a component of the organization's business activities and is used to support strategic business choices.
- The appointing party creates the document.
- It should include topics such as risk assessment and management, capital investment and lifecycle costing, environmental management, etc.,

The OIR contributes to the **Project Information Requirements** (UK BIM Framework, 2022b):

- The PIRs make it possible to comprehend the high-level data that the appointing party needs when working on a design and construction project.
- Before any appointments for consultants or contractors are made, they must be finished.
- Also, the appointing party creates it.
- It should incorporate topics such as corporate key performance indicators, Project business cases, Strategic briefs, etc.

Then the PIR contributes to the **Exchange Information Requirements** (UK BIM Framework, 2022b):

- The information that must be provided by a lead appointed party or by an appointed party at each information exchange should be specified in the EIRs at a level that can be written into contracts.
- Before each appointment, it must be defined and provided as part of the hiring procedure.
- It can be created by the appointing to the lead appointed party or by the leader to the appointed party.
- The EIR should include the structuring, definition, and purposes of the information that is being requested.

It's crucial to supply structured information to achieve open, shared data, and the EIR must outline the material's structure. As seen in Figure 4, this would entail providing the structured data which shall be exported for initial purposes and fed into secondary needs. Therefore, the major purposes should be taken into consideration before the subsidiary purposes while considering the structured information's contents. Following the completion of the initial goal, the data can be transferred as structured or unstructured, according to the subsequent goal.

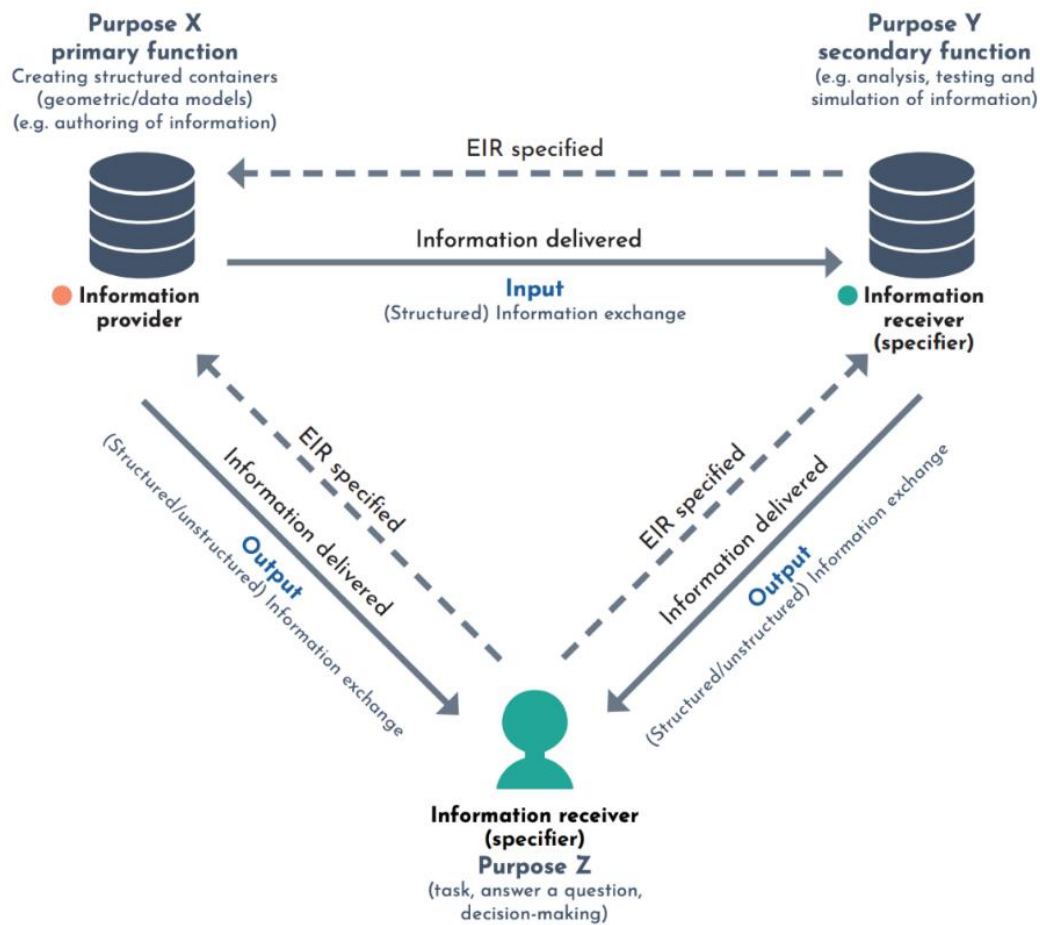


Figure 4: Exchange Information Requirements and Information Delivery (UK BIM Framework, 2022b)

### 2.1.2 Level of Information Need

The Level of Information Need is a framework meant to define the quality, quantity, and granularity of information requirements. It should be used to establish clearly and communicate the requirements of well-defined purposes of a project, being accurate with the amount of information need (UK BIM Framework, 2020).

This framework has an important impact on defining structures to require and deliver the information, in such a way that it enables the development of automatic compliance checks of the deliverables. Figure 1 illustrates the process of defining the structure and granularity of information needs, providing a skeleton of information containers, and each one should be the result of one or multiple information requirements defined for each purpose.



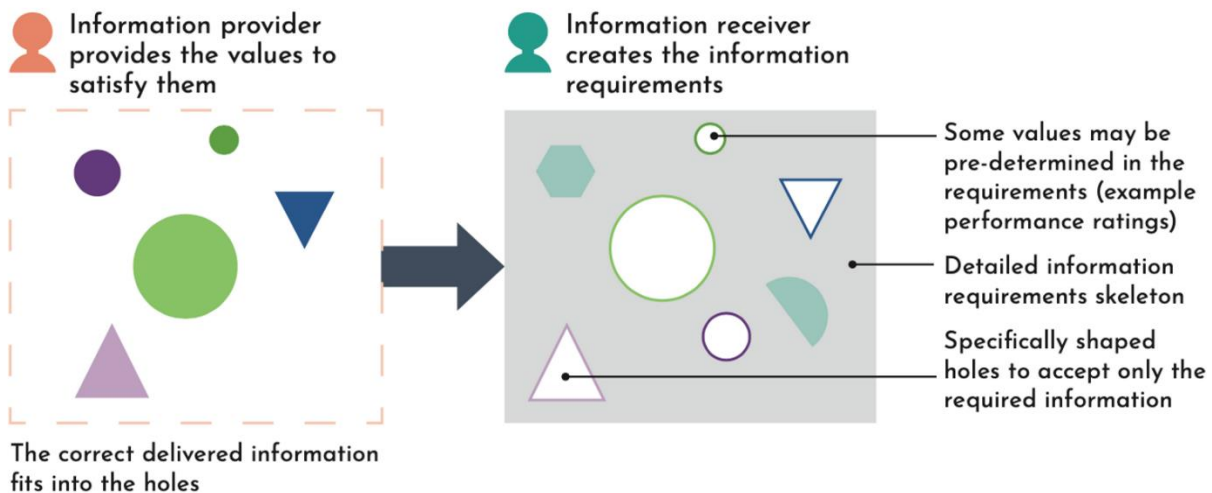


Figure 5: Information Requirements and Delivery (UK BIM Framework, 2022b)

The Level of Information Need framework has three sub-divisions:

- Geometrical information,
- Alphanumerical information.
- Documentation.

For each sub-division, it must be attached the purpose of the information that will be generated, the milestones, the responsibility, and the granularity.

The Level of Information Need must be defined every time that an information requirement is produced, which can be in the OIR, PIR, AIR, or EIR. In this way, it should be used by the Appointing Parties, in any of those documents, and, by the Lead Appointed Parties, as an example, defining the level of information need during the preliminary design phase to perform accessibility analysis (UK BIM Framework, 2020).

## 2.2 Open BIM Workflows

Open BIM is a paradigm shift that fosters collaborative processes and frictionless data exchange in the building sector. Its emphasis on open standards and non-proprietary formats promotes interoperability among different BIM software products, allowing architects, engineers, contractors, and other stakeholders to collaborate successfully throughout the construction lifecycle.

The International Alliance for Interoperability, buildingSMART, is a critical driver of Open BIM adoption. As a global non-profit, it actively develops and supports open standards and tools, which serve as the foundation of Open BIM processes, such as:

- Industry Foundation Classes (IFC), responsible for information transport.

- Model View Definitions (MVD), responsible for the translation of processes into requirements.
- BIM Collaborate Format (BCF), responsible for coordination processes.
- Information Delivery Manual (IDM), responsible for describing processes.
- buildingSMART Data Dictionary (bSDD), responsible for the definition of BIM objects and attributes.
- Information Delivery Specification (IDS), responsible for structuring information requirements.

The benefits of using open data standards can include aspects such as more transparent, collaborative, and open workflows, greater information certainty due to a shared vocabulary of industry terms, greater reuse of data, and easier integration with linked data created and shared in related industries. Jiang et al., produced a systematic analysis of openBIM standards, with publications from several databases, from 2000 to 2018, which showed the growing interest of the industry in Open standards, especially IFC. The work concludes that openBIM has been playing a great role in the AEC industry, promoting effective solutions to interoperability problems, and facilitating the collaboration process of projects.

In this research, the focus will be on the IFC and IDS standards, which will be base for the main methods of the Marketplace.

### **2.2.1 IFC – Industry Foundation Classes**

The IFC standard includes definitions that cover data required for buildings over their life cycle, being an open standard for Building Information Model (BIM) data that are exchanged and shared (ISO 16739-1,2018). The documentation of the schema is composed of an EXPRESS schema specification and alternatively, an XML schema specification, and reference data, represented as XML definitions of property and quantity definitions.

Software applications are required to support well-defined subsets of the data schema and referenced data. These subsets are referred to as a Model View Definition MVD (buildingSMART International, 2017). At the moment, the official version of IFC Standard is IFC4\_ADD2\_TC1 - 4.0.2.1, which supports two base MVDs:

- Reference view: with the objective is to support software and workflows that do not require modifying geometry. Such as coordination planning, clash detection, background reference, quantity take-off, construction sequence, and visual presentation.
- Design transfer view: Advanced geometric and relational representation of spatial and physical components to enable the transfer of model information from one tool to another.

Each MVD represents a subset of the IFC Schema, which the conceptual architecture is represented in Figure 6. As mentioned, several workflows can be covered by these base MVDs, identifying data exchange requirements that should be supported by the conforming software applications.

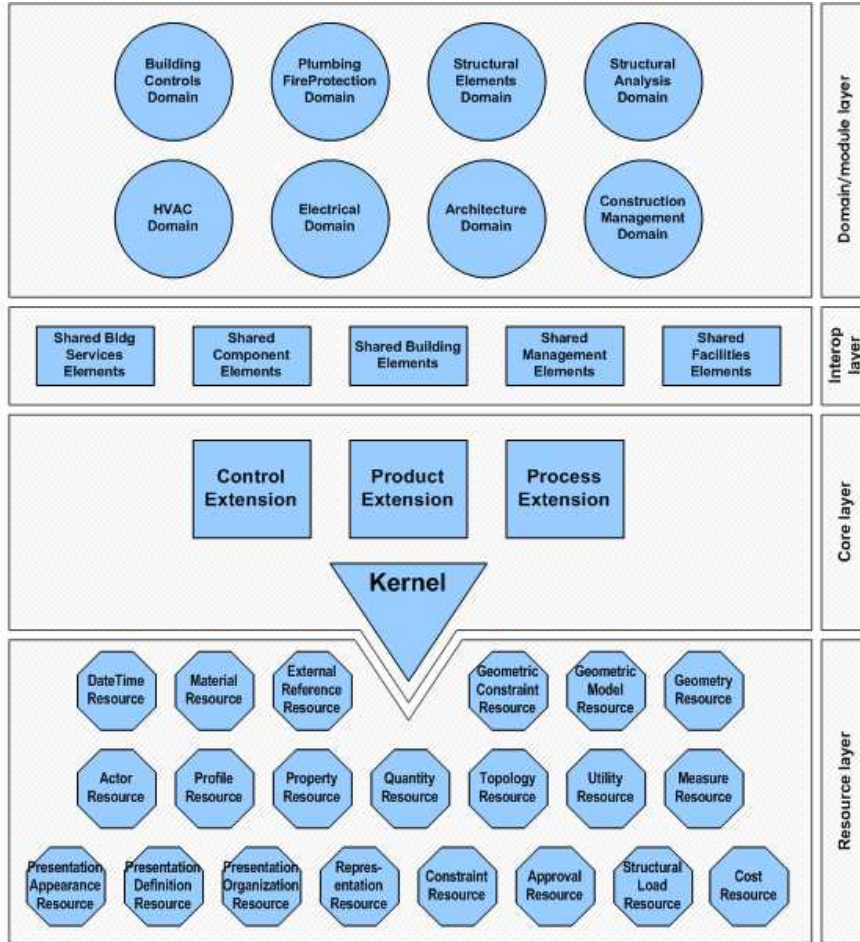


Figure 6: IFC 4.0.2.1 Data Schema Architecture (buildingSMART International, 2017)

As illustrated in the figure, the Schema has four main layers, defined by official documentation:

- **Resource layer:** includes all individual schemas containing resource definitions, such as Actor, Cost, Material, Approval, and many others. Those definitions do not include a globally unique identifier and shall not be used without being nested to another declared entity of another layer, in this way, having the related objects to those resources.
- **Core layer:** this layer includes the kernel schema and the core extension schemas, which contain the general entity definitions. All entities of this layer, or above carry a globally unique ID.
- **Interoperability layer:** this layer includes definitions that are utilized for inter-domain exchange and sharing of construction information, such as Shared Building Elements like Walls, which can be used in Architectural or Structural Domains.
- **Domain layer:** the highest layer includes schemas containing specialized definitions of entities for discipline/domain-specific uses.

The IFC Standard has three more official formats: “.ifcZIP”, data embedded into a zip file, which is encoded in EXPRESS or XML; “.itf” (Terse Triple Language) and “.rdf” (Resource Description

Framework), which both are based on the ifcOWL ontology. The ifcOWL ontologies are generated directly from the IFC Express Schema, a process highlighted in Figure 7.

| IFC Schema                   | ifcOWL Ontology                              |
|------------------------------|--|
| Simple data type             | owl:class + owl:DatatypeProperty restriction |
| Defined data type            | owl:class                                    |
| Aggregation data type        | owl:class                                    |
| SET data type -----          | ----- non-functional owl:ObjectProperty      |
| LIST & ARRAY data type ----- | ----- indirect subclass of express:List      |
| Constructed data type        | owl:class                                    |
| SELECT data type -----       | ----- rdfs:subClassOf for owl:classes        |
| ENUMERATION data type -----  | ----- rdf:type for owl:NamedIndividuals      |
| Entity data type             | owl:class                                    |
| Attributes -----             | ----- object properties                      |
| Derive attr                  | -  |
| WHERE rules                  | -  |
| Functions                    | -  |
| Rules                        | -  |

Figure 7: IFC EXPRESS to ifcOWL (buildingSMART International, 2016)

Pauwels and Terkaj (2016) defined an EXPRESS to OWL conversion pattern that can be implemented in several different algorithms, Figure 8. The work resulted in two parallel converters, one using Java and the other C++, and both resulted in identical *ifcOWL* ontologies, which proves the reproducibility of the defined pattern. To instate the *ifcOWL* ontologies is recommended to develop an ontology-based application to automate the procedures of parsing and generation of individuals. Figure 8, is highlighted as a graph representation for the *IfcPropertySetDefinitionSets*.

```

ifc:IfcPropertySetDefinitionSet
  rdf:type owl:Class ;
  rdfs:subClassOf
  [
    rdf:type owl:Restriction ;
    owl:allValuesFrom ifc:IfcPropertySetDefinition
    ;
    owl:onProperty express:hasSet
  ] ;
  rdfs:subClassOf
  [
    rdf:type owl:Restriction ;
    owl:minQualifiedCardinality "1"^^xsd:
      nonNegativeInteger ;
    owl:onProperty express:hasSet ;
    owl:onClass ifc:IfcPropertySetDefinition
  ] .
    
```

Figure 8: Printout of the RDF graph representation for the *IfcPropertySetDefinitionSet* (Pauwels & Terkaj, 2016)

Another possible format for the IFC Schema is the JSON, JavaScript Object Notation, although it is not an official format as the others mentioned previously, it is classified as Provisional/Candidate by buildingSMART. The *ifcJson* proposition has appeared intending to improve the performance of web services, which work with data transmission between applications, when compared to the XML data serialization of IFC (Afsari et al., 2017).

The implementation developed by Afsari et al. of *ifcJSON4* is an interpretation from the IFC specification constrained by the JSON schema. The methodology used for the data model mapping is divided into three main steps:

- *ifcJSON4* Schema development through the IFC4 EXPRESS source and the JSON Schema specification.
- *ifcJSON* document implementation, a JSON document meant to be validated against the *ifcJSON4*.
- Data validation approach which addresses the validation approaches.

Figure 9 can be seen as a brief representation of the proposed schema.

```

ifcJSON4 Schema
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "title": "ifcJSON4 Schema",
  "description": "This is the schema for representing IFC4 data in JSON",
  "definitions": {
    "description": "This includes select IFC datatypes.",
    "ifcValue": {},
    "ifcAxis2Placement": {}
  },
  "type": "object",
  "properties": {
    "description": "This includes all IFC entities",
    "ifcOwnerHistory": {},
    "ifcProject": {}
    "ifcCartesianPoint": {},
    "ifcAxis2Placement2D": {},
    "ifcAxis2Placement3D": {}
  },
  "required": ["ifcOwnerHistory", "ifcProject"]
}

```

Figure 9: Brief representation of the overall ifcJSON4 schema structure(Afsari et al., 2017).

### 2.2.2 IDS – Information Delivery Specification

The ISO 19650 brought a more dynamic approach to Exchange Information Requirements (EIRs) which mixed up with the concept of Model View Definitions (MVDs), which were well defined with standard constraints, requirements, and subsets of the IFC schema. Considering this dynamic approach, the continuous implementation of new MVDs to follow up with not standardized EIRs is not sustainable, since these EIRs can define requirements not covered by the MVD subset, or even outside of the IFC

schema (Berlo, 2019). This condition is illustrated in Figure 10, in which the orange rectangles represent the EIR's scopes, the green circles the MVDs subset, and the blue shape the IFC schema.

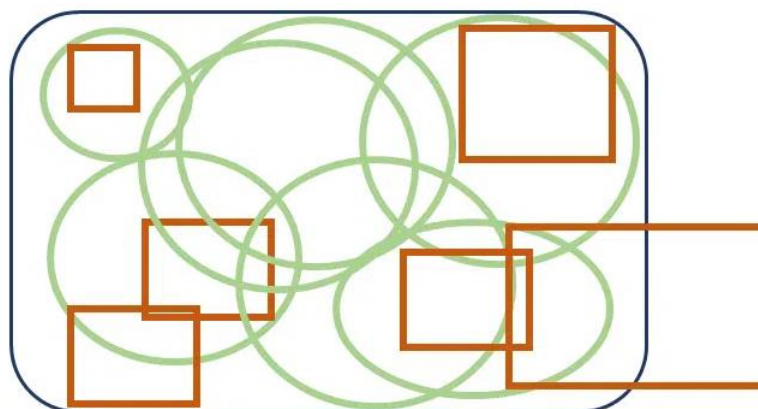


Figure 10: The expansion of MVDs and EIRs adding data from outside an MVD (Berlo, 2019)

In this context, **buildingSMART** started to develop a new format, called Information Delivery Specification (IDS), which is a computer-interpretable document that defines the Exchange Requirements, structuring the project requirements based on the objects, classifications, and properties. In this way, it can be a combination of Domain Extensions, IFC entities, and additional standardizations from national and company agreements. Therefore, the IDS file can be used to define the Level of Information Needs of use-specific requirements and, ensure that all the required information is provided, enabling Clients to check whether the IFC model meets all of the defined Specification. A macro flow of this process can be seen in Figure 11.

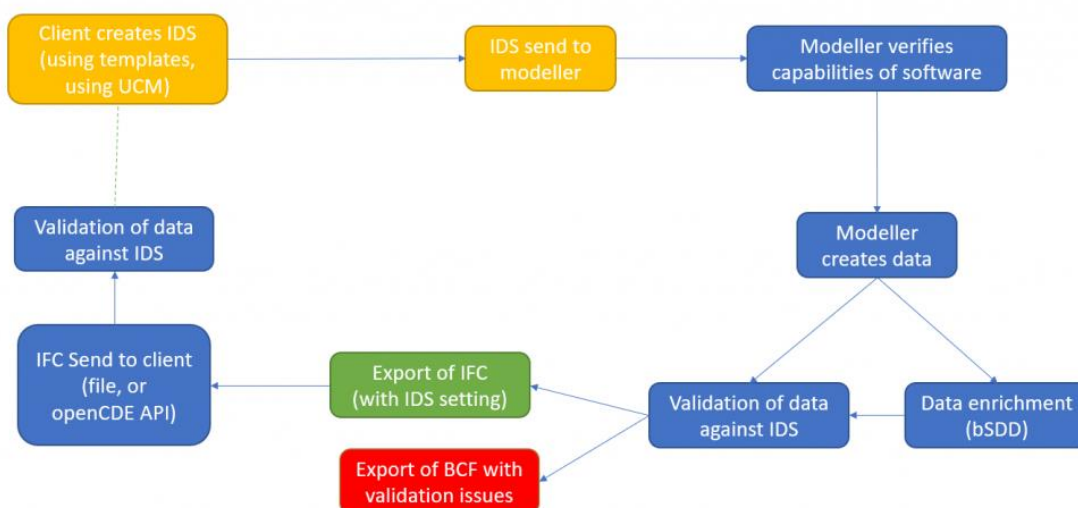


Figure 11: IDS macro flow (buildingSMART International, 2020)

In the IDS documentation, buildingSMART presents the file structure as a list of specifications, and each specification is defined by three main parts:

- **Description:** explanation of the specification, intended for humans to read and understand the motivations of the requirements.
- **Applicability:** define the scope of the application, a subset of objects.
- **Requirements:** define the information required from the objects that apply to the specification.

The Applicability and Requirements are structured using Facets, which is a method to describe the information using fixed Parameters, enabling computers to interpret the data precisely. In the Applicability part, a Facet has the objective of describing information that points to a set of objects, and in the Requirements, it describes what kind of information those objects should have. There are six different types of Facets, each one with its fixed Parameters, which can be seen in Table 1.

Then, the Facet Parameters can be specified by simple values or complex restrictions, such as a list of values, a range of numbers, or regular expressions. All this structure is formulated in tags of an XML format. Each Specification or Requirement can also be defined as **Required**, **Optional**, or **Prohibited**, using the tap properties “*minOccurs*” and “*maxOccurs*”. In Figure 12 it is possible to observe the general structure in an example of an IDS file.

In comparison between standardized and non-standardized methods to specify information requirements in digital construction projects, IDS was identified as the most advantageous for automated compliance checking for validation of alphanumerical information (Tomczak et al., 2022) For this review the following methods were selected:

- Product Data Templates (PDT),
- Model View Definition (mvdXML).
- Information Delivery Manual (IDM).
- Level of Information Need, Data Dictionaries (ISO12006).
- IFC Property templates.
- Information Delivery Specification (IDS).
- Linked Data with SHACL.
- non-standardized textual or spreadsheet documents (DOC & XLS).
- proprietary solutions, such as Solibri.
- other such as dedicated visual programming scripts.

**Table 1:** List of Facets and its Parameters (Adapted from buildingSMART International, 2021)

| Facet Type     | Facet Parameters                     | Example applicability  | Example requirement  |
|----------------|--------------------------------------|--|--|
| Entity         | IFC Class (Name) and Predefined Type | Applies to "IfcWall" with a predefined type of "SHEAR"   | Must be an "IfcWall" with a predefined type of "SHEAR"                                 |
| Attribute      | Name and Value                       | Applies to elements with the attribute "Name" having the value "W01"                                     | Must have the attribute "Name" with the value "W01"                                    |
| Classification | System and Value                     | This applies to elements classified under "Uniclass 2015" as "EF_25_10_25"                               | Must have a "Uniclass 2015" classification reference of "EF_25_10_25"                  |
| Property       | Property Set, Name, and Value        | Applies to elements with a property set of "Pset_WallCommon" with a "LoadBearing" property set to "TRUE" | Must have a "Pset_WallCommon" property set with a "LoadBearing" property set to "TRUE" |
| Material       | Value                                | Applies to "concrete" elements   | Must have a "concrete" material  |
| Parts          | Entity and Relationship              | Applies to elements that are "contained in" an "IfcSpace"  | Must be "contained in" an "IfcSpace"   |

```

<ids:info>
  <ids:title>IDS Test Gabriel Veloso</ids:title>
  <ids:description>Information Requirements from BIMA</ids:description>
</ids:info>
<ids:specifications>
  <ids:specification ifcVersion="IFC4" name="Classification System Columns" minOccurs="1">
    <ids:applicability>
      <ids:entity>
        <ids:name>
          <ids:simpleValue>IFCCOLUMN</ids:simpleValue>
        </ids:name>
      </ids:entity>
    </ids:applicability>
    <ids:requirements>
      <ids:classification minOccurs="1" maxOccurs="1">
        <ids:value>
          <ids:simpleValue>Pr_20_85_16_80</ids:simpleValue>
        </ids:value>
      </ids:classification>
    </ids:requirements>
  </ids:specification>
</ids:specifications>
</ids:ids>
    
```

Figure 12: Snapshot of an IDS file



### 2.3 BIM Data transfer methods and parsing

There are several workflows for BIM data transfer already established, considering property formats (e.g. Revit Files) or open formats (e.g. IFC). Pauwels et al., grouped three of the most promising approaches. Each transfer method can be defined as more fine-grained decisions and options according to the requirements and needs of the use cases.

The first group of methods is the traditional IFC-based file transfer. This group is broken down into three approaches, the first (TM1.1), is based on the step format of IFC, it starts with the definition of the correct MVD for the case, to be able to embrace the data requirements for the use. Depending on which system kind of system the data will be transferred, a parser such as *ifcOpenShell* will be required. The second approach is to use the RDF file version of IFC (TM1.2), which usually will require a converter of the step version to the RDF, a process that can make constantly the RDF graphs outdated. The last (TM1.3), is to use the JSON format of IFC, which also relies on a conversion, which can be done with several tools, such as Blender, and, with the IFC.js library.

The second group of methods is related to live-linked data servers and is divided into two. The first (TM2.1), consists of an IFC server, which would require an automated conversion process of IFC to RDF in the backend of a server, storing the RDF graph in a triple store. The second (TM2.2), is the Linked Building Data (LBD) server, which requires a process where the BIM Model Data is directly converted into RDF LBD graphs, and those are inserted into RDF-based triple stores.

The last group is equally web-based as the linked data server approaches in TM2. Although, they rely on JSON-based web services. The TM3.1 consists of a web service that returns JSON data of the BIM Model, considering a custom server that responds to HTTP requests. The JSON representation is not well standardized but could be used in a custom structure such as the IFC.js. For the TM3.2 approach, a connection to live web-based APIs is required, assuming that the BIM data will be streamed directly from the common data environment (eg. BIM360) (Pauwels et al., 2023).

|  | Standardization | Transfer steps | Dependencies       | Extensibility | Reliability |
|--|-----------------|----------------|--------------------|---------------|-------------|
| <b>Traditional IFC-based file transfer</b> |                 |                |                    |               |             |
| TM1.1 - SPF and MVDs                       | Broad           | Few            | Software-neutral   | Limited       | Outdated    |
| TM1.2 - IFC-RDF File transfer              | Broad           | Many           | Software-neutral   | Medium        | Outdated    |
| TM1.3 - IFC-JSON File transfer             | Narrow          | Many           | Software-dependent | High          | Outdated    |
| <b>Live Linked Data Server</b>             |                 |                |                    |               |             |
| TM2.1 - IFC Server                         | Broad           | Few            | Software-neutral   | Medium        | Outdated    |
| TM2.2 - LBD Server                         | Medium          | Medium         | Software-neutral   | High          | Up-to-date  |
| <b>JSON-based web services</b>             |                 |                |                    |               |             |
| TM3.1 - Retrieve web-based JSON data       | Narrow          | Few            | Software-neutral   | High          | Up-to-date  |
| TM3.2 - Connection to live web-based API   | Narrow          | Few            | Software-dependent | High          | Up-to-date  |

Figure 13: Comparison between each data transfer method (Pauwels et al., 2023)

Considering, the use case of the conducted research, “Live semantic data from building digital twins for robot navigation”, and the summary of the comparison, the author, selected to implement the methods TMI.3 – IFC-JSON File transfer and TM2.2 – LBD Server with the object to get an indication of the constrains, limits and possibilities with each method.

The use of the **IFC.js library** is mentioned as a possible part of data transfer workflows, especially TM1.3 and TM3.1. This library can enable the generation of 3D scenes being compatible with 3D libraries such as Three.js or Babylon.js and promote access to all the properties associated with that geometry, such as their materials, thermal characteristics, structural strength, etc. (IFCJS, 2022). IFC.js is composed of 3 main layers: **web-ifc**, **web-ifc-three**, and **web-ifc-viewer**.

**Web-ifc details (IFCJS, 2022):**

- **Description:** an IFC file parser, which can read all information from an IFC, write on files, and save.
- **When it should be used:** when it is only necessary to manipulate the IFC data, reading or writing in the files without a 3D viewer, as shown in **Table 2**.

**Table 2:** Examples of Methods from the Web-ifc API

| Method and Arguments   | Functionality   |
|--|---|
| <i>IfcAPI.openModel(data: Uint8Array, settings: LoaderSettings): number</i>        | Opens a model and returns the <i>ModelID</i> .            |
| <i>IfcAPI.CreateModel(settings: LoaderSettings): number</i>                        | Creates a new model and returns the <i>ModelID</i> .      |
| <i>IfcAPI.GetGeometry(modelID: number, geometryExpressID: number): IfcGeometry</i> | Get mesh geometry using <i>ModelID</i> .                  |
| <i>IfcAPI.ExportFileAsIFC(modelID: number): Uint8Array</i>                         | Exports a file using the <i>ModelID</i> .                 |
| <i>IfcAPI.WriteLine(modelID: number, lineObject: any)</i>                          | Write a line using <i>ModelID</i> and <i>lineObject</i> . |

**Web-ifc-three details (IFCJS, 2022):**

- **Description:** a 3D BIM viewer that allows one to view and navigate the 3D model. It is the official IFC Loader of three.js. Through the *IFCLoader* object, the load functions are accessible as any other loader.

- **When it should be used:** when it is required an IFC 3D viewer for the application, with full control over the viewer functionalities, and access to all model information, as described in **Table 3**.

**Table 3:** Examples of Methods from the Web-ifc-three API

| Method and Arguments   | Functionality  |
|--|--|
| <i>async IfcLoader.IfcmManager.getAllItemsOfType (modelID: number, type: number, verbose: boolean): number[]   object[];</i> | Returns all objects of the specified IFC type.                 |
| <i>async IfcLoader.IfcmManager.getItemProperties(modelID: number, id: number, recursive = false): object[];</i>              | Gets the native properties of the given element.               |
| <i>async IfcLoader.IfcmManager.getPropertySets(modelID: number, id: number, recursive = false): object[]</i>                 | Gets the property sets and quantity sets of the given element. |
| <i>async IfcLoader.IfcmManager.getSpatialStructure (modelID: number):</i>  | Gets the spatial structure of the project.                     |
| <i>IfcLoader.IfcmManager.createSubset(config: SubsetConfig ): object</i>   | Creates a new geometric subset.                                |

**Web-ifc-viewer** details (IFCJS, 2022):

- **Description:** A 3D BIM viewer with many tools and functionalities already implemented.
- **When it should be used:** when it is possible to use pre-made functions to optimize the model navigation tools implementations, such as section drawings, dimensions, annotations, etc. Some examples are highlighted in **Table 4**.

**Table 4:** Examples of Methods from the Web-ifc-viewer API.

| Method and Arguments  | Functionality   |
|---|---|
| <i>IfcViewerAPI.IFC.loadIfcUrl(url: string, fitToFrame: boolean, onProgress: any onError: any)</i>          | Load the Ifc from a <i>URL.createObjectURL</i> and add it to the scene. |
| <i>IfcViewerAPI.grid.setGrid(size: number, divisions: number, colorCenterLine: Color, colorGrid: Color)</i> | Sets the grid on the viewer.  |
| <i>IfcViewerAPI.IFC.selector.pickIfcItemsByID(modelID: number, ids: number focusSelection: boolean)</i>     | Highlights the item with the given ID with the picking material.        |

Donkers A. and Yang, developed a tool called LBDviz which combines the use of the IFC.js library to generate a browser-based viewer and semantic web technologies to link other kinds of data outside of the IFC scope. The tool was tested in multiple use cases, such as checking design requirements, generating design feedback, and maintaining building performance in a sustainable and economically viable manner. The graphical user interface, based on IFC.js, serves as a gateway to interact with the decentralized knowledge graphs.

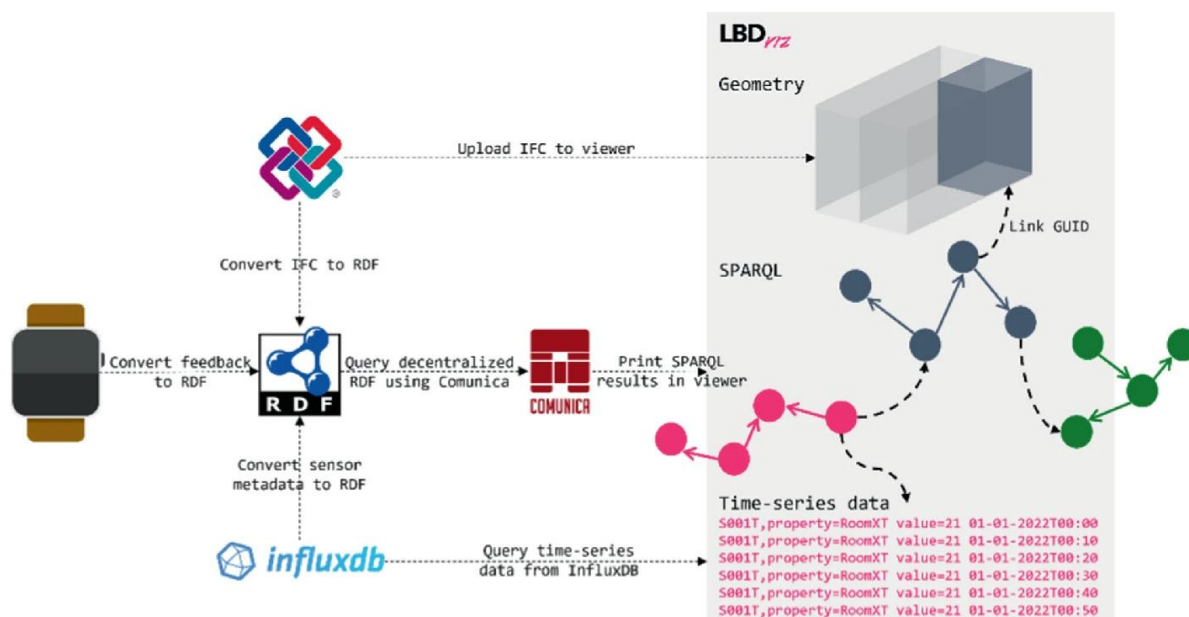


Figure 14: LBDviz system architecture (Donkers A. and Yang, 2023)

## 2.4 Web3 Technologies in the AEC Industry

While the deployment of Blockchain and other WEB3 technologies has already benefited sectors like finance, banking, and supply chain, the construction sector has yet to establish itself. The foundation for many recent innovations has been the fusion of AEC technology, like as BIM, with Web3 services. The use of smart or self-executing contracts operating on blockchain to minimize costs and time in projects before, during, and after construction can be supported by these kinds of connections. These concepts include tying physical construction to its digital counterpart on this platform. BIM-enabled industry advancements toward deeper levels of collaboration necessitate increased stakeholder integration, transparency, and a distinct separation of duties, which can be leveraged by Blockchain technologies (Erri Pradeep et al., 2019).

Some relevant works towards the use of Web3 technologies in the AEC Industry are being developed, one of them is **BIMchain**. It generates digital evidence of various BIM workflow transaction scenarios and stores it on a public blockchain like Ethereum. By creating a Deliverables Management System that incorporates BCT into the regular flow of information interchange between stakeholders on existing BIM platforms, BIMchain is also working to enhance BIM workflows. Additionally, it sees the use of

smart contracts in the future to facilitate payments and a decentralized project cloud for information sharing via a peer-to-peer blockchain secured protocol (Erri Pradeep et al., 2019).

Various research has implemented smart and secure construction applications using the Internet of Things, Artificial Intelligence, Cloud-to-Edge computing, Blockchain, and Digital-Twin. Kochovski and Stankovski (2021) offer a study that explores issues connected to the development, deployment, and operation of AI-based building systems. They looked into information collection, fusion, and enrichment techniques that can provide intelligence during the building process and help to better a variety of the necessary tasks. Both time-sensitive activities like collision detection and early disaster warning, as well as longer-term logistical and other operations like buying supplies, recording the process, and so on, can benefit from these collections of information on the current building process. Four use-case scenarios were taken into consideration for implementation in the study:

1. Increase workplace safety by sending notifications to the site manager.
2. Monitoring the movement of vehicles into and out of the building site.
3. Resources, waste, and assets management.
4. Working conditions monitoring the environmental conditions at the construction site.

The most relevant project for this ongoing research is BUILDCHAIN, already introduced as a Horizon Europe-founded project, that aims to be a digital solution for improving building life cycle knowledge. The objective is to create a marketplace where various actors may share their offers, including their quality certificates and credentials, and where all activities linked to the full life cycle of buildings can be tracked. The EU-funded BUILDCHAIN project will create the Digital Building LogBook (DBL), which municipalities will use to maintain and govern their building inventory. The Decentralized Knowledge Graph (DKG) open-source blockchain-based solution will be used to integrate existing and new data, tools, and functions into the DBL. The program will feature building-specific ontologies that trace and continually update the life cycle of buildings, following twelve mapped use cases:

- **U1:** Life cycle analysis and carbon footprint computations.
- **U2:** Reasoning architecture for the management of a large population of buildings.
- **U3:** Structural Health Monitoring (SHM), reliability assessment, and anomaly detection with AI sensor fusion
- **U4:** Earthquake and climate-proof building based on digital twinning.
- **U5:** SHM tools for cultural heritage buildings.
- **U6:** Follow-up procedures of design processes by Bayesian updating.
- **U7:** Precipitation monitoring for active control of local flooding.
- **U8:** Optimized 'self-attentive' sensing.
- **U9:** Post-catastrophic interventions

- **U10:** Construction economics
- **U11:** Management of deep renovation workflows.
- **U12:** Increase operational energy and energy efficiency of buildings.

The **BUILDCHAIN** project is the main reference for the development of the work being presented here, especially, considering use case 10, about construction economics. In this way, the following sub-sections will dive into concepts that will leverage the development of the proposed marketplace.

#### **2.4.1 Blockchain and Decentralized Applications**

At its most basic level, blockchain is a set of accepted guidelines for managing a database. These guidelines create a dependable, secure, and consistent record that permits data interchange and validation between and by a wide range of stakeholders. Even if the parties do not trust one another, the protocols that control data security and dependability provide a tamper-proof record that serves as the basis for trust in the information. (Greenwald et al., 2020).

Every event is recorded as a "block" in a chronological "chain" that is time-stamped, attached to the identity of the individual who produced the encrypted data, and cannot be edited once created. Without verification, a "block" cannot be added to the chain. It cannot be modified once added. A blockchain "ledger" is a record that is formed by a chain of blocks. Blockchain technology produces a "single source of truth."(Greenwald et al., 2020). Some of Blockchain technology's main characteristics are:

- **Cryptography** – Blockchain uses cryptographic techniques for two main purposes: ensuring the immutability of records and securing the identity of transaction senders using public and private keys, creating a chain of validated blocks with cryptographic hashes, and preventing any modification to data once included in the chain.
- **Decentralized P2P Network** – the blockchain is not stored in only one place, it is a shared database, which actors can view, add, and validate nodes in the chain.
- **Nodes** – the computers connected to the blockchain.
- **Distributed Ledger** – ledger refers to the blockchain transaction, hence, A geographically dispersed database of digital data that has been copied, shared, and synced is known as a distributed ledger.
- **Validity Rules and Consensus Mechanisms** - The blockchain rules determine how information is validated, and when it is validated, the consensus of all participants determines whether the verified block should be added to the chain and in what order.

Building trustworthy decentralized applications (**DApps**) is not direct since most blockchain-based functionality must be designed from scratch and integrated with data semantics. Hyperledger Fabric, Ethereum, R3 Corda, Tezos, EOSIO, and Ontology are just a few of the blockchain frameworks that

have been proposed to provide versatile and adaptive solutions for various applications. These platforms enable the execution of smart contracts, which are self-executing applications capable of reading and writing the state of the blockchain on which they are realized (Papaioannou et al., 2022). Papaioannou et al. (2022), proposed a new software framework, named **ONTOCHAIN**, to leverage blockchain-based and semantically enriched trustworthy decentralized applications. It consists of an innovative protocol suite made up of lower-level core protocols like authorization, certification, privacy-aware data processing, cross-chain gateways, identity management, secure data exchange, and high-level application protocols like data provenance, reputation models, decentralized oracles, market mechanisms, ontology representation, and management. On top of this framework, a DApp would be deployed, such as semantic digital logbooks for companies, buildings, trustworthy logistics, and supply chains.

#### 2.4.2 Self-Sovereign Identities

Self-sovereign identification (SSI) is a new model for online digital identity: how we verify who we are to websites, services, and apps with which we need to build trustworthy connections in order to access or secure private information. SSI, like earlier technological paradigm shifts, is driven by new technologies and standards in encryption, distributed networks, cloud computing, and smartphones. On a higher level, SSI is a collection of technologies based on fundamental concepts in identity management, distributed computing, blockchain or distributed ledger technology (DLT), and cryptography (Preukschat et al., 2021). The model can be broken down into seven building blocks as follows:

- Verifiable credentials (digital credentials)
- The trust triangle: issuers, holders, and verifiers
- Digital wallets
- Digital agents
- Decentralized identifiers (DIDs)
- Blockchains and other verifiable data registries
- Governance frameworks (trust frameworks)

Figure 15 depicts the three main roles involved in the exchange of verifiable credentials using the terminology established by the W3C Verifiable Claims Working Group:

- **Issuers** are the source of credentials. Every credential has one. The majority of issuers are entities such as governments, financial institutions, universities, organizations, and so forth. Individuals, as well as things, can be issuers. For instance, a sensor with the appropriate hardware may produce a digitally signed certificate for a sensor reading.

- **Holders** obtain Verifiable credentials from issuers, store them in their digital wallet, and provide claims verification using one or more of those credentials when verifiers request it and the holder accepts it.
- **Verifiers** can be any person, group, or entity looking to establish credibility regarding the subjects of credentials. Holders of one or more claims from one or more Verifiable credentials are required to show proof in order to be verified. The verification of the issuer's digital signature, which is commonly achieved via a **DID**, is a critical stage in this procedure.

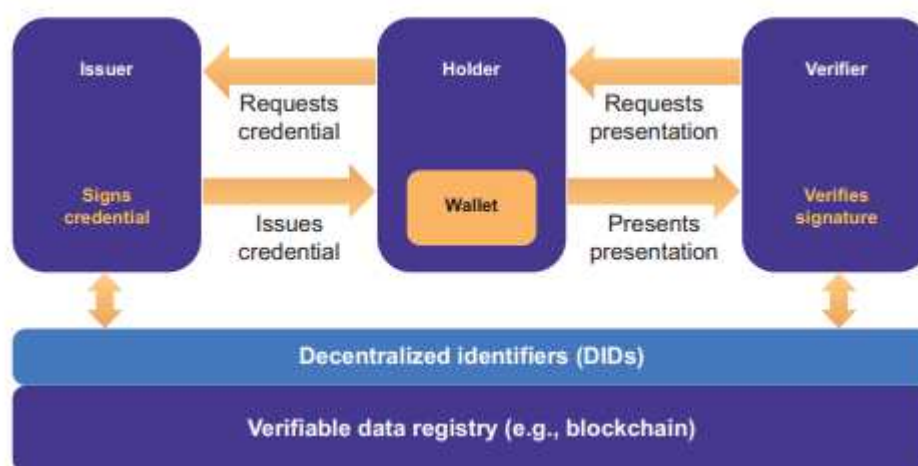


Figure 15: The primary roles involved with the exchange of verifiable credentials (Preukschat et al., 2021)

It should be noted that the trust triangle only describes one side of a business transaction. In many business transactions, both sides ask the other for information. As a result, in a single transaction, both parties serve as holders and verifiers. Furthermore, many business transactions result in the issuance of a new credential from one party to the other, or even two new credentials, one in each direction (Preukschat et al., 2021).

Diving into the DID concept, it is a URI that can be a URL or a URN and can be looked up to obtain a standardized set of information about the resource specified by the DID. If the recognized resource has one or more web representations, one or more of those URLs can be included in the metadata, Figure 16, it is illustrated the main schema and relationships of the concept. DIDs have four primary properties (Preukschat et al., 2021):

- An identifier that is permanent (persistent) never has to update.
- A resolvable identifier that can be used to find metadata.
- A cryptographically verifiable identification that can be used to demonstrate control.
- A decentralized identifier – No need for a centralized registration authority.



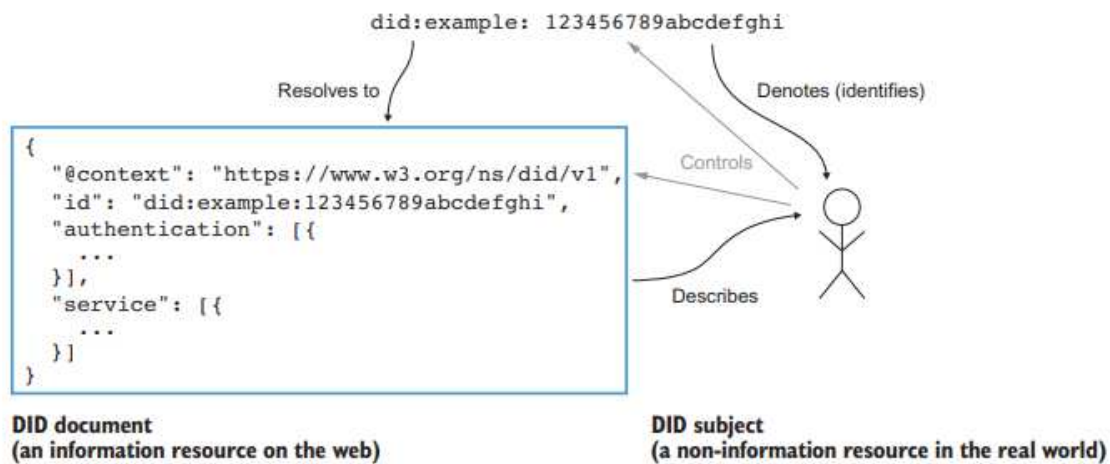


Figure 16: Relationships between the DID, DID document, and DID subject (Preukschat et al., 2021)

Cocco et al., offered a model of information management for the construction industry that makes use of blockchain, IoT, and BIM technology as well as self-sovereign identity ideas. The data model enables the identification of all interested parties, the storage of all information off-chain, and the use of multi-signature approval processes when necessary. Only the data necessary for information notarization and certification is stored on-chain.

The system proposed by the authors involves three subsystems, each with its own actors and features: Blockchain subsystem, IoT device subsystem, and Information management platform. The actors, elements outside of the system, that interact with the tools, provide inputs, and trigger actions, are divided into five groups: the **System administrator** that manages the deployment on the Blockchain of the contracts; **DIDs** are the Decentralized Identifiers representing building artifacts; **Smart Contracts** are solely the ones who interact with the Blockchain without intermediation; **IoT Devices**; and, the **information management system** during the whole cycle of a building asset (Cocco et al., 2022).

Figure 17, shows a High-Level UML (unified model language) class diagram describing the designed solution. It is composed of seven classes:

- **Creator/Admin**: the person who implements the external system and contracts.
- **Entity**: class designating a DID-below natural person, legal person, or object.
- **DID**: class representing stakeholders and construction products.
- **Information Management Platform**: that communicates with the DID and the Admin actor.
- **IoT Device**: class for IoT equipment owned by a DID.
- **Notarized File**: It has data that can be used to establish who is responsible for the entities.
- **Verifiable File**: A certified organization issues this kind of file, which includes data used to support characteristics or features.

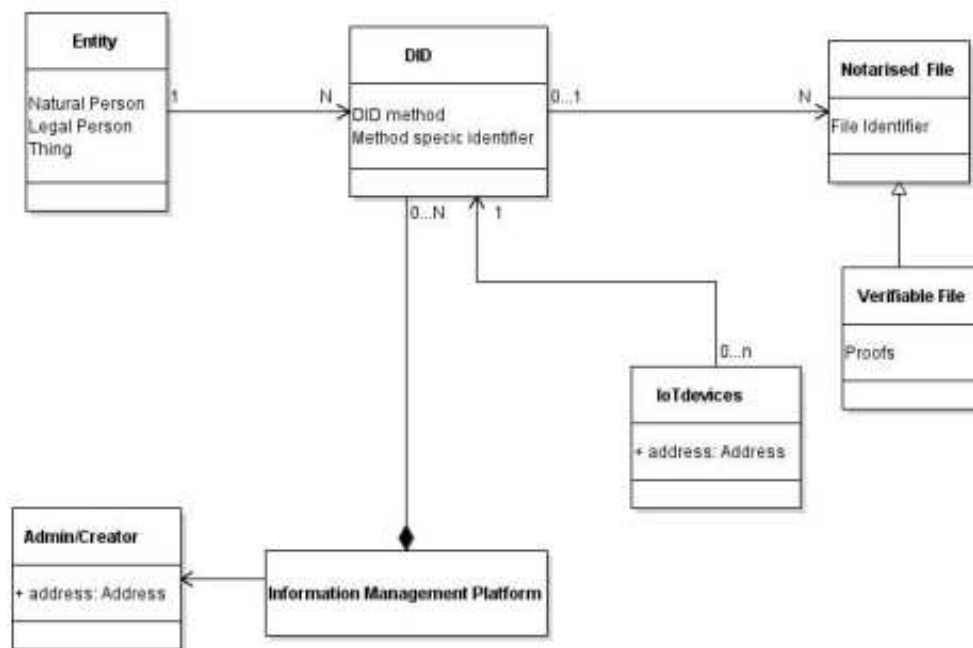


Figure 17: Class Diagram of the system (Cocco et al., 2022)

### 2.4.3 NFT – Non-Fungible Tokens

Non-Fungible Tokens (NFTs) are smart contract standards that can be used to tokenize real-world assets. Tokens are digital assets that belong to a single user and are safely stored on the blockchain. They are based on a set of rules recorded in a smart contract. Teisserenc and Sepasgozar (2022) proposed a software architecture and leveraged the non-fungible token standard to develop a framework for smart contracts that addressed the key use cases for Digital Twins in the Built Environment. Among several problems identified and addressed in the study, decentralization, and automation of tendering, notarization of the financial supply chain, payment automation, and the creation of digital assets through tokenization, incentivization methods, data ownership, and IP protection, and pricing data authentication, are especially relevant in this context.

There are other relevant novel approaches using NFTs in the AEC Industry, which aim to support BIM models and their automatic checking. It can be used in a way to reflect unique attributes achieved by a certain design team, being built, and shared on the blockchain network, becoming accessible and consultable by all, giving trust to this team during tenders and appointments (Pattini et al., 2022). It also can be addressed by utilizing blockchain technology as a tool for authorship and/or ownership attribution of a specific file, when NFTs are an excellent possibility for the digital design sector because they were created specifically to handle the ownership of digital assets in a secure (Casillo et al., 2022).

#### 2.4.4 Decentralized Knowledge Graphs

Although, this concept was briefly introduced in section 2.3 about **BIM Data transfer methods and parsing**, in the LBDViz system architecture (Donkers A. and Yang, 2023). The last topic of this review is meant to exemplify how Decentralized Knowledge Graphs can support data compliance in DApps within AEC Domains with trustworthy BIM data.

Several papers discuss the integration of BIM data with knowledge graphs and blockchain, but, in real-world situations is quite common that these data are not 100% correct, and the lack of metadata correctness checking inevitably compromises BIM data quality in the blockchain. Tao et al., proposed a knowledge graph-driven smart contract (KGSC) for metadata checking to enhance input quality in a blockchain-based BIM collaboration. Then, the proposed smart contract algorithm of this work has two stages:

- **Generation of BIM metadata checking rules (MCR).** The ISO 19650 standards have recommended BIM cooperation techniques, including data quality management systems. However, this knowledge is dispersed and, in some cases, ingrained. As a result, knowledge graph (KG) technology is used to extract information about BIM information sharing. Semantic Web Rule Language (SWRL) is also used to mine embedded knowledge and build MCRs.
- **Development of the KGSC algorithm.** A new KGSC algorithm will be developed by integrating MCRs.

Therefore, this work contributes to 29 rules that are reasoned and generated from the ISO 19650 knowledge graph. The results indicate that BIM data processed by the KGSC has achieved higher compliance and accuracy (Tao et al., 2021).

### 3 METHODOLOGY

This section outlines the systematic approach to fulfill the research objectives, with a focus on the integration of Building Information Modeling and Web3 services within the Architecture, Engineering, and Construction business. The technique is divided into three parts: the first will be the gathering of requirements collected in the Literature Review, based on the analysis of the BUILDCHAIN objectives; the second define the user stories of the use case development for the decentralized marketplace, leveraging the ISO 19650 workflows; and then, the detailed design of the solution is presented considering its data model and architecture, then, divided into work items to develop using an Atomic Agile Methodology.

#### 3.1 Requirements Gathering and Analysis

Considering the several applications and processes reviewed in section 2, and the basis of BUILDCHAIN decentralized knowledge graph, the following requirements were gathered to build a scalable application in its full potential:

- **An IFC Parser and IFC-RDF Converter:** BIM data will serve as the marketplace's foundation by providing the necessary digital representation of construction projects, including data about buildings, services, and stakeholders. Also, to support the IDS checking and subsets generation, to deliver only the required information.
- **Web User Interface:** Web application that provides the front-end interfaces for clients and stakeholders to access the marketplace. The main requirement for this interface is to support an IFC viewer. It will facilitate user interactions, service assignments, reputation systems, and transparent communication.
- **Knowledge Graph:** The Knowledge Graph database will be used to store interconnected data from BIM models. It will facilitate seamless information interchange and data interoperability by enabling efficient querying, data linking, and semantic representation.
- **Blockchain Services:** will secure the market's data's security, transparency, and immutability. Smart contracts will be used to automate transactions, enforce agreements, and govern the lifetime of NFT-based services, defined by the IDS subsets.
- **Decentralized Identifiers (DIDs):** DIDs will be utilized to provide self-sovereign identities to various stakeholders in the marketplace, including Clients, Contractors, Service Providers, and objects. Each entity will have control over its identity, ensuring secure authentication and privacy.
- **Reputation System:** A reputation system to be integrated into the marketplace to track and assess Contractors' and Service Providers' performance and dependability. It will employ Client

feedback, ratings, and reviews to build trust and encourage high-quality service delivery, correlating to the NFTs and DIDs.

- **Infrastructure and Deployment (Docker):** Docker containers can be utilized to install and scale the marketplace's components, ensuring flexibility and easy deployment across several settings.

### 3.2 Use Case Development for the Decentralized Marketplace

To develop the study case of this research, the user stories for this novel approach of the Decentralized Marketplace were refined considering the eight project stages and the three actors defined in ISO 19650, to ensure the compliance of the application with a well-defined workflow. In the following figures, it can be seen business process is modeled with these considerations. In Figure 18, the full process is represented, and in the others zoom into specific steps.

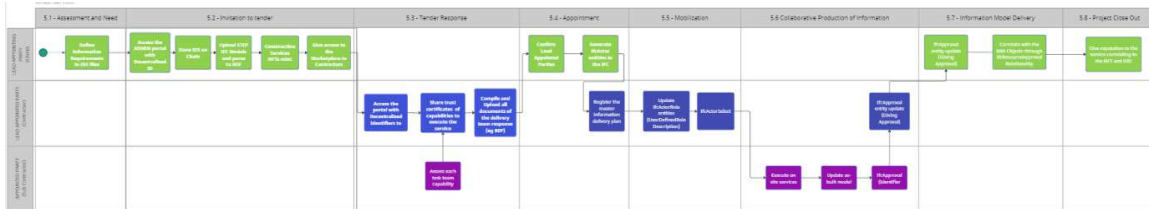


Figure 18: Business Process Model

The first two stages, 5.1 – Assessment and Need and 5.2 – Invitation to Tender, are related to processes restricted to the Client, Appointing Party, when it is put high focus on defining project requirements, organizing documents, to make the call to the Tender. In Figure 19, this process is mimicked in a simplified way, considering the scope of the Application. In stage 5.1 the client defines the Requirements in a IDS files, to have a structured format for this kind of data. Then, in 5.2, he accesses the web application with his Decentralized Identity, to store the requirements on-chain and upload/connect the IFC Models that are the reference datasets for the construction services. The IFC parser functions developed with IFC.js will process these files, and then, connect with the Blockchain Services, to generate the data subsets, considering the IDS file. The NFTs are minted reflecting the construction services that each of these subsets represents. To close this process, the Appointing Party invites the contractors to the Marketplace.

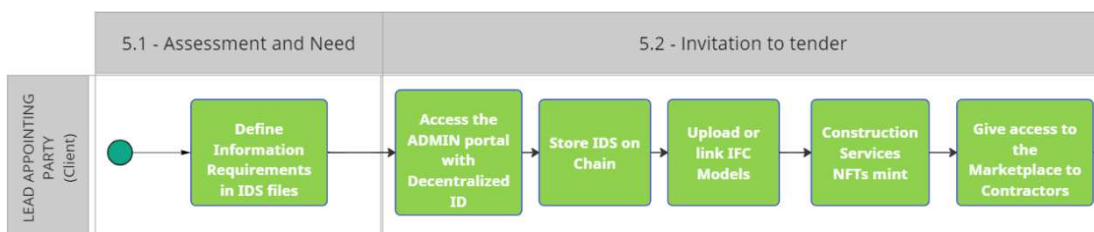


Figure 19: Zoom into the Stages 5.1 and 5.2

In Figure 20, the stages of Tender Response and Appointment are represented and the interaction with the two other players, the Lead Appointed Party and Appointed Party starts. It begins with the Lead Appointing Party accessing the Marketplace with his Decentralized Identity, connecting his Wallet that contains the proofs of his capabilities for the available work that his aiming for. Then, he will upload the pre-appointment documents like the BIM Execution Plan (BEP) to the platform.

In 5.4, the Client confirms the Lead Appointed Party based on the documentation provided, which in the backend service will integrate the Decentralized Identities of the Delivery Team with *IfcActor* entities in the IFC file and transfer the ownership of an NFT. Then, the Lead Appointing Party will provide the final delivery plan, based on the Exchange Information Requirements.

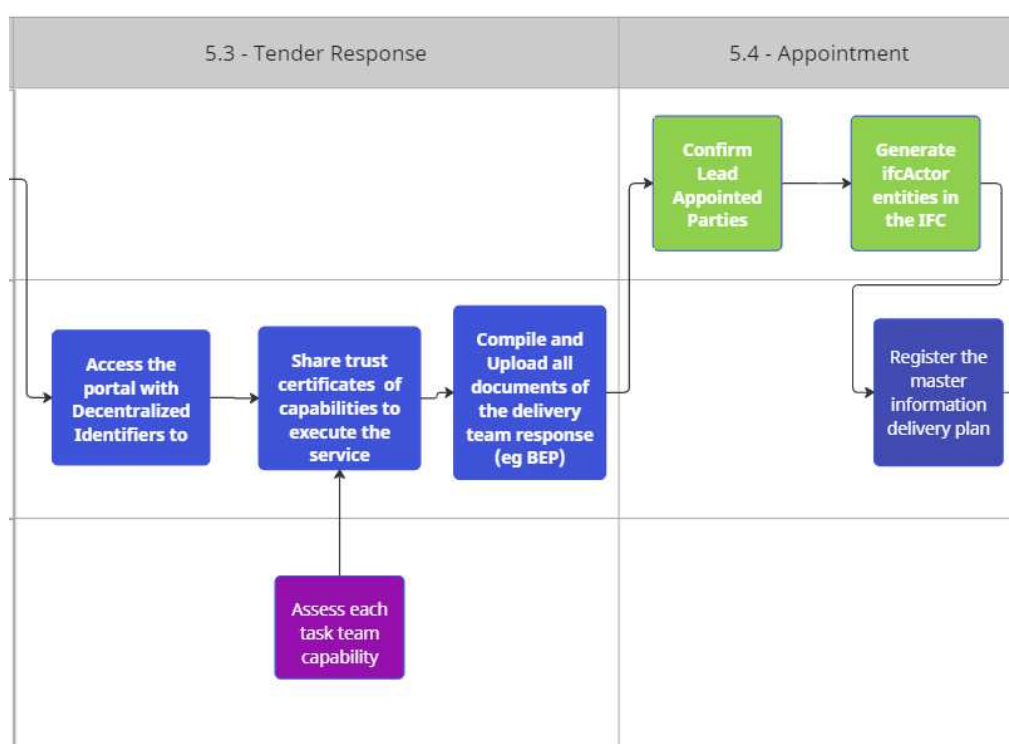


Figure 20: Zoom into the Stages 5.3 and 5.4

In the Mobilization phase, through the web application, the Lead Appointing Party will update the *IfcActor* entities with the *IfcActorRole*, especially with the Appointing Party interactions. The Collaborative Production of Information is the stage in which the Appointing Party has a higher focus. Since, this study case deals with Construction Delivery, in this stage, the focus is on the execution in the site and updates of as-built models. At this point, the Appointing Party can provide DID documents to prove the properties of the materials used on site and correlate this with the BIM Objects within the knowledge graph service (The structuring of DID Documents is not a scope of this study case). To close this phase, the Appointing Party will require Approval through the *IfcApproval* entity connecting to the BIM Objects related to the scope of the service, and the Lead Appointing Party will start the revision process, as shown in Figure 21.

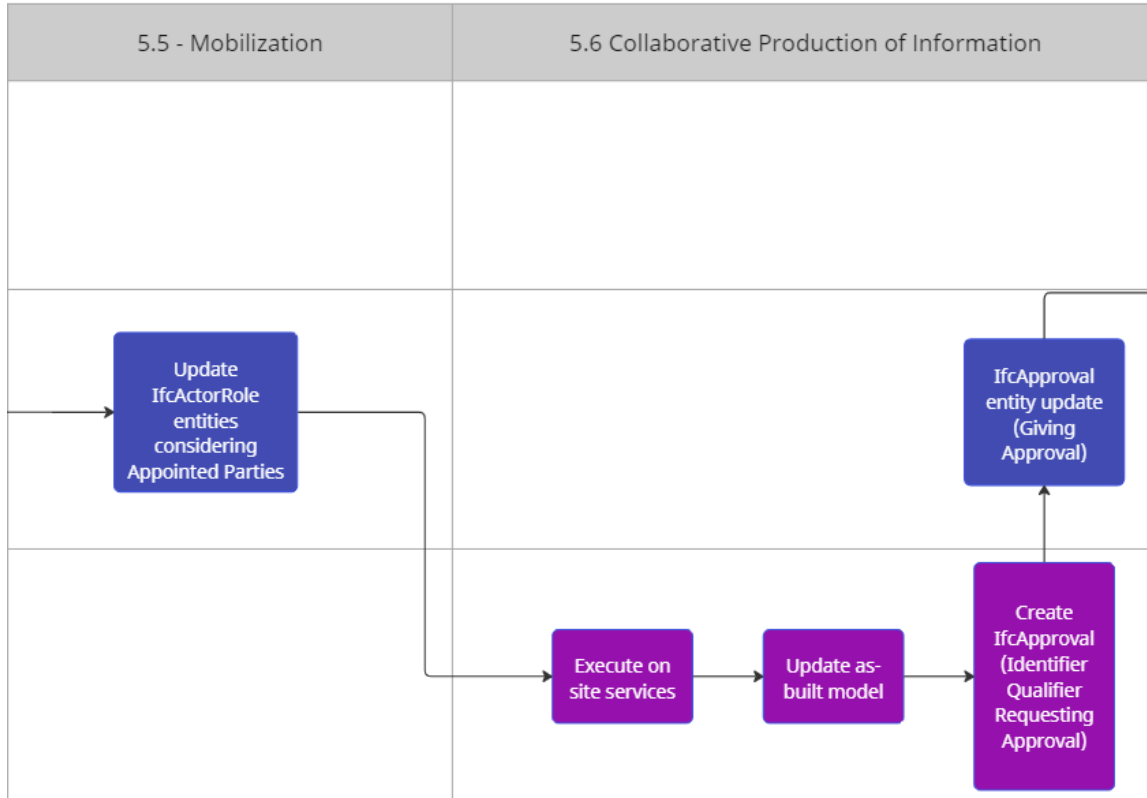


Figure 21: Zoom into the Stages 5.5 and 5.6

Then, to confirm the delivery, the Approval process will be pointed out to the Client. Based on the service execution, through the Reputation System service, the Client will provide a score for the job corresponding to the initial NFT as shown in Figure 22. This reputation can be also attached to the Delivery Team DID which will recursively be used to prove capabilities in future work bids.

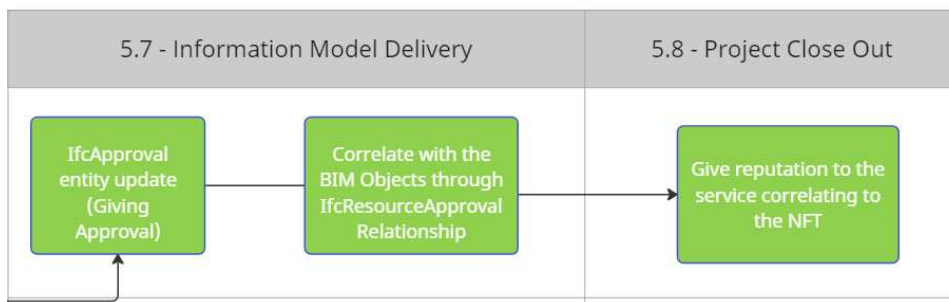


Figure 22: Zoom into the Stages 5.7 and 5.8

### 3.3 Detailed Design of the Solution

To cover all the defined user stories, a complex system architecture is required, that shall be composed of a frontend to enable all the users' interactions and, some backend services to process the data parsing, exchange, and storing. The services connections are illustrated at a high level in Figure 23.

In this way the frontend will be responsible for:

- Handle user interactions.
- Parse and display IFC data in 3D viewers with the IFC.js framework.
- Print query results on the Knowledge Graph.
- IDS parser to deserialize the data, check IFC, and generate subsets.

And the backend services will be:

- NFT minter.
- Smart contract to handle the transactions on the marketplace.
- IFC Approval to RDF.
- DID Documents link.

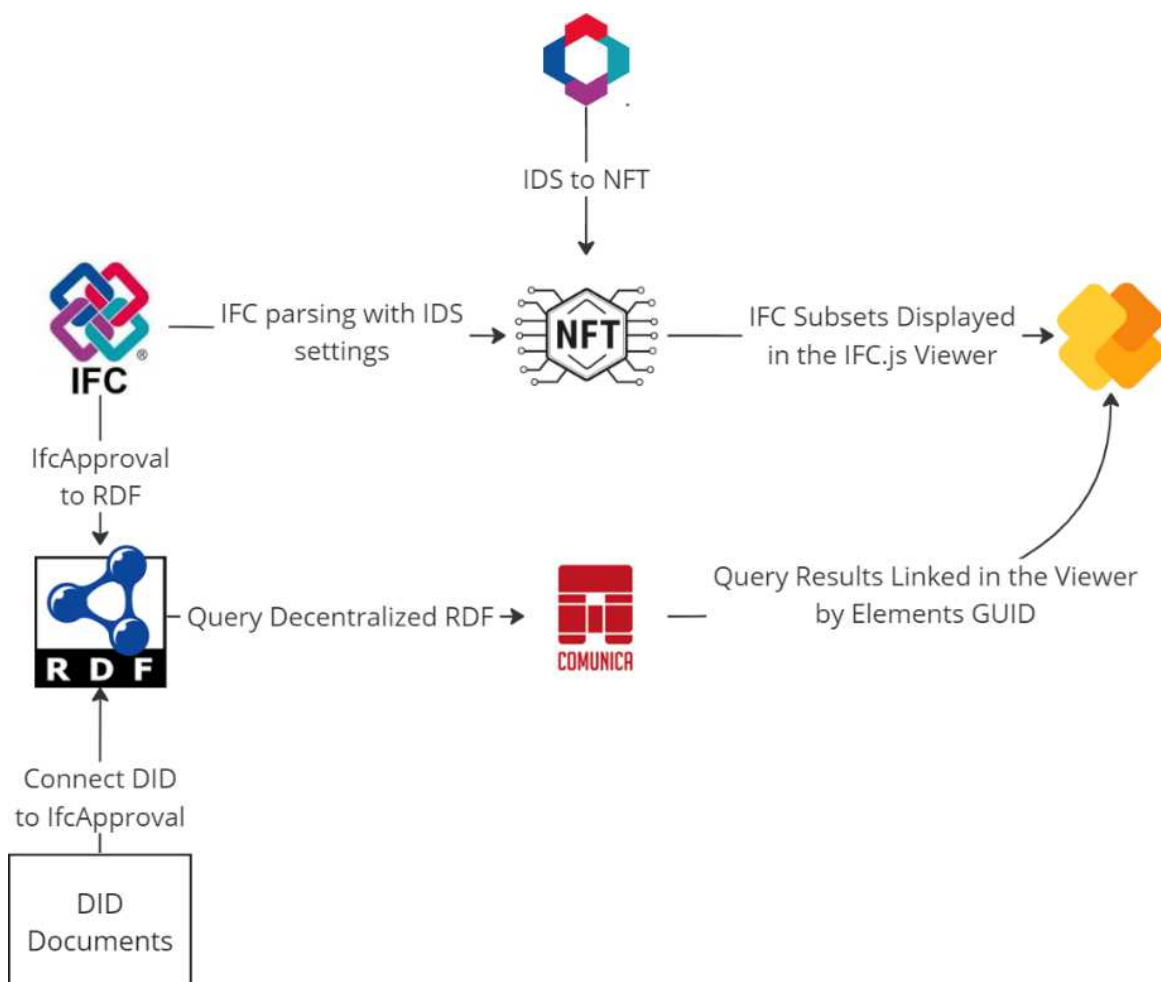


Figure 23: System Architecture

The Atomic development approach seeks to define the smallest sizes of viable coding tasks, which means user stories that cannot be broken down, in order to deliver stable functionalities for a system. Considering the scope of this study case, all the development tasks are presented in **Table 5** in the



implementation order, since a correct sequence enables achieving faster Minimum Valuable Products that can anticipate results and highlight new needs. The full implementation and demonstration are discussed in the next section.

**Table 5:** Atomic packs of development

|    |  |
|----|--|
| 1  | Implementation of IFC 3D Viewer  |
| 2  | IDS entity facet methods   |
| 3  | IDS attribute facet methods  |
| 4  | IDS classification facet methods                                       |
| 5  | IDS material facet methods   |
| 6  | IDS property facet methods   |
| 7  | IDS partOf facet methods   |
| 8  | IFC subsets definition   |
| 9  | <i>NFTActor</i> (Smart Contract)                                       |
| 10 | <i>OpenBIM_D</i> Actor (Smart Contract)                                |
| 11 | Function of Mint NFTs  |
| 12 | Function to Transfer Ownership of NFT                                  |
| 13 | Function to Request Approval of Service                                |
| 14 | Function to Give Approval and Reputation to Service                    |
| 15 | Function to write <i>IfcOrganization</i> entities inside the IFC model |
| 16 | Function to write <i>IfcActorRole</i> entities inside the IFC model    |
| 17 | Function to write <i>IfcActor</i> entities inside the IFC model        |
| 18 | Function to write <i>IfcApproval</i> entities inside the IFC model     |
| 19 | Functions to write relationships between entities inside the IFC model |
| 20 | Frontend for the Marketplace page                                      |
| 21 | Frontend for the Minter page   |
| 22 | Frontend for the “MyNFTs” page   |
| 23 | DID integration to <i>IfcApproval</i>                                  |
| 24 | RDF query functionalities  |
| 25 | Queries display on frontend  |

### 3.3.1 Data Model

In Figure 24 it is presented the High-Level ontology for this implementation, which highlights the main actors, objects, and properties of the proposed application. Each group has a specific responsibility to sustain. The arrows represent the communication direction between these elements.

The Smart Contracts (NFTs as well) are responsible for making the bridge between the off-chain services and the blockchain, connecting the IFC data displayed with the IDS data on the chain, or even, connecting the approval processes with the reputation system. The *OpenBIM\_D* Actor is a Smart Contract responsible for handling the interactions in the Marketplace with the users, keeping track of the data, and calling the *NFTActor* when is needed to Mint an NFT or to transfer its ownership. The *NFTActor* is the Smart Contract for the NFT itself.

The *IFCParser* is set a of functions developed with IFC.js to manipulate all the processes related to the IFC files. Therefore, the Smart Contracts interact with these functions to achieve the goals of generating the subsets according to the IDS and writing *IfcActor* and *IfcApproval* entities. These are native entities in the IFC Schema, and they are highlighted in the figure because they will play key roles in storing information in the model. The DID Document is a simplified element to represent the mechanisms to establish verifiable credentials of the execution of the construction services, thus, they will interact with the *IfcApproval*.

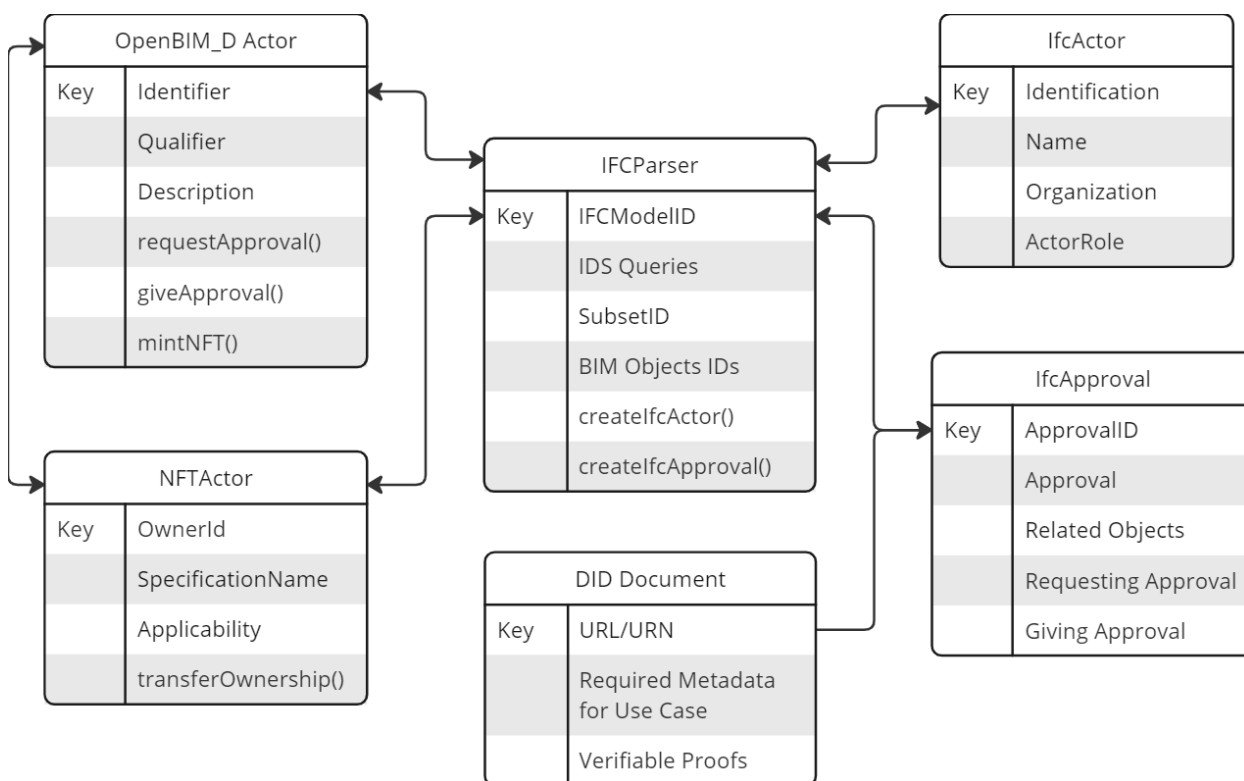


Figure 24: High-Level Ontology

## 4 IMPLEMENTATION AND DEMONSTRATION

This chapter will be broken down into a brief explanation about the technology stack used, then, the actual implementation of each step, demonstrating its outputs and integrations. Thus, the last section will bring the perspective of each kind of user while using the application during different stages of the use case process.

### 4.1 Technology Stack and Tools

To implement this complex system, the set of technologies, tools, and programming languages is diverse. Although, four elements are keys to this implementation:

- **IFC.js:** the functionalities of this library were already extensively discussed in section 2.3. This library is the basis of the implementation of items 1 to 8 and 15 to 19, from **Table 5**, since IFC.js will be used to cover all IFC data parsing, generation of the 3D viewer, and writing new information inside the IFC models.
- **Internet Computer Blockchain-based Network:** Internet Computer (IC) is a revolutionary new blockchain design that unleashes the full potential of smart contracts by overcoming the limits of smart contracts on standard blockchains in terms of speed, storage costs, and computational capability. For the first time, smart contracts can construct fully decentralized applications that are hosted end-to-end on the blockchain. The IC is a collection of cryptographic protocols that link independently operated nodes into a network of blockchains (Shoup, 2022). Another relevant point to choosing IC is the low cost of on-chain storage, which nowadays is around **5 dollars per year for 1GB**, which makes an extremely considerable difference when compared to Ethereum, which costs around 240,000,000 dollars per year. This characteristic makes sustainable and scalable the possibility to store the IDS files on chain. Therefore, this technology was selected to implement 9 to 14 from **Table 5**.
- **React:** React JS is a JavaScript package used to create visually appealing user interfaces. Because of its totally component-based architecture, it is emerging as one of the quickest and easiest frontend libraries to utilize in online applications (Maratkar & Adkar, 2021). React is being used to implement the frontend of the DApp, which is referred to in items 20 to 22 in **Table 5**.
- **Comunica:** is a modular knowledge graph querying framework for JavaScript that fits the requirements of being a flexible query engine that permits plugging in new components such as various algorithms, new or experimental SPARQL features, and support for new Web interfaces (Taelman et al., 2018). This JavaScript package is present in the implementation of items 24 and 25 from **Table 5**.

## 4.2 Integration of IFC.js viewer and IDS (Information Delivery Specification)

This section covers the implementation of the items 1 to 8 from the Table 2. Since the Atomic methodology suggests the continuous delivery of Value, these functions were developed to also work without the other steps, as it will be possible to observe at the end of the section. For this implementation, it was necessary to use two layers of the IFC.js library: **web-ifc-viewer** and **web-ifc-three** (IFCJS, 2022). The first is responsible for generating the viewer itself and getting the spatial structure of the model, and the second is to parse and obtain the attributes, properties, and materials data from the elements, relationships between entities, resources, and classification systems.

The first step of this implementation is to instantiate the viewer, and then load the IFC Model. For this, the container (HTML DIV) is gotten in order to use *IfcViewerAPI*, to generate the viewer at the position of the selected “DIV”. As represented in Figure 25, the viewer is generated twice, one will be responsible for holding the whole IFC Model, and the other, just the subset defined by the IDS file. This structure is thought to clearly highlight the subsets of data, to check if the IDS parser functions created are working according to expected.

```
1 const viewer_container = document.getElementById('viewer-container');
2 const viewer_containerIds = document.getElementById('viewer-container-ids');
3 function createViewer(viewer_container){
4     const container = viewer_container;
5     const viewer = new IfcViewerAPI({ container, backgroundColor: new Color('#E2F0D9') });
6     const manager = viewer.IFC.loader.ifcManager;
7     viewer.grid.setGrid();
8     viewer.axes.setAxes();
9     return viewer;
10 };
11 const viewer = createViewer(viewer_container);
12 const viewerIds = createViewer(viewer_containerIds);
```

Figure 25: Generation of the IFC.js viewer

With the method “*loadIfcUrl*”, the IFC model is loaded by passing its URL. Then, the express ids of all elements in the Spatial Structure of the IFC (*IfcProduct*, *IfcSite*, *IfcBuilding*, *IfcBuildingStorey*, *IfcSpace*, and many other entities derive from *IfcProduct*) are stored to be used in following functions. This process is highlighted in Figure 26.

```
1 const ifcURL = URL.createObjectURL(ifcFile);
2 modelIds = await viewerIds.IFC.loadIfcUrl(ifcURL);
3 const ifcProject = await viewerIds.IFC.getSpatialStructure(modelIds.modelID);
4 flattenedExpressIds = getAllExpressId(ifcProject);
```

Figure 26: Loading the IFC Model

After this point, the functions implemented to parse the IDS file and generate the respective subsets, according to the Applicability and Requirements take place, all the objects and functions were developed considering the 0.9 version of the IDS Schema. The first step is to read the XML file and store its data into objects, which will simplify handling the hierarchy of the Specifications, Applicability, Requirements, Facets, and Parameters. The structure that is demonstrated in **section 2.2** of this document. In Figure 27, the “**IDS\_Specification**” class is defined, this is the class responsible for storing the top-level data of the specifications, having in its constructor the arguments of the specification’s name and occurrence, and the applicability and requirements that are defined by following objects.

```
1 export class IDS_Specification {
2   constructor(Name, Applicability, Requirements, Occur) {
3     this.Occur = Occur;
4     this.Name = Name;
5     this.Applicability = Applicability;
6     this.Requirements = Requirements;
7   }
8 };
```

Figure 27: Specification Object

Figure 28 represents the “**IDS\_Specification\_Definition**” class, which has the responsibility to define the Applicability and Requirements objects, storing their occurrence and the facets that compose them.

```
1 export class IDS_Specification_Definition {
2   constructor(Facets, Occur) {
3     this.Occur = Occur;
4     this.Facets = Facets;
5   }
6 };
```

Figure 28: Specification Definition Object

The “**IDS\_Facet**” class represented in Figure 29 has the key objective to store the facet type (Entity, Attribute, PartOf, Properties, Classification, and Material), and the facet parameters that are being used.

```
1 export class IDS_Facet {
2   constructor(Parameters, FacetType) {
3     this.Parameters = Parameters;
4     this.FacetType = FacetType;
5   }
6 };
```

Figure 29: Facet Object

The last class created to parse the IDS data is the “IDS\_Facet\_Parameter” which is defined to store the data from the lowest layer of the IDS Schema, the facet parameters. Each facet has a specific set of parameters that can be used, in order to, associate with specific information in the IFC Models, therefore the parameter name is a key resource. The restriction will tell how the value of the parameter might be used, being from a simple value or going in the direction of complex restrictions, regular expressions are a possible type of complexity. This class is displayed in Figure 30.

```
1 export class IDS_Facet_Parameter {
2   constructor(ParameterName, Restriction, ParameterValue) {
3     this.ParameterName = ParameterName;
4     this.Restriction = Restriction;
5     this.ParameterValue = ParameterValue;
6   }
7 };
```

Figure 30: Facet Parameter Object

Then, this set of four classes can cover the whole complexity of the IDS schema, allowing to store data from multiple specifications, when each one can have a diversity of applicability and requirements, which might have different facets with several different parameters. The combination of two functions that were developed do the work of getting the data from the IDS file and transposing to the objects created from these classes, making available the information in a much easier way to be reused while the application is running. The first function “getSpecificationData” has the responsibility to obtain the arguments to generate the “IDS\_Specification” object, in this it gets the name of the specification and calls the next function which will obtain the definitions for the applicability and requirements. The process is represented in Figure 31.

```
1 export function getSpecificationData(specificationIDS, specificationsParsed){
2   const specName = specificationIDS["$"]["name"]
3   const applicability = getSpecificationDefinition(specificationIDS, "ids:applicability");
4   const requirements = getSpecificationDefinition(specificationIDS, "ids:requirements");
5   const specificationIDSObj = new IDS_Specification(specName, applicability, requirements);
6   specificationsParsed.push(specificationIDSObj);
7 }
```

Figure 31: Method to parse the Specification Data

The *getSpecificationDefinition()* method iterates through parts of the IDS, to get the from the parameters and facets layers, to construct the applicability or the requirement object. As it can be seen in Figure 32,

each “for” iteration goes to a different layer of the schema to obtain the specific data, which will dynamically compose the specifications definitions.

The next step of the implementation was to create the IFC query functions for each facet type. These functions are responsible for parsing the IFC schema with the IFC.js library methods, according to the parameters obtained for each facet during the IDS parsing. Therefore, each function is specialized in only one type of facet, but the general process is the same, runs across the parameters and returns an array of element IDs which will be used to create the subset of data.

All the facets were covered, although not all the possible parameters are fully covered, and only the “simpleValue” condition is implemented, which means that the complex restrictions, at this moment are out of the scope.

```

1  export function getSpecificationDefinition(specificationIDS, applicabilityOrRequirements){
2      const applicability = specificationIDS[applicabilityOrRequirements];
3      const facetsApplicabilityName = Object.keys(applicability);
4      let facets = [];
5      for(const facetName of facetsApplicabilityName) {
6          const facet = applicability[facetName];
7          const parameterNames = Object.keys(facet);
8          let parameters = [];
9          const facetObj = new IDS_Facet(parameters, facetName.replace("ids:", ""));
10         facets.push(facetObj);
11         for(const parameterName of parameterNames) {
12             const parameter = facet[parameterName];
13             const parameterRestriction = Object.keys(parameter)[0];
14             const parameterValue = parameter[parameterRestriction];
15             const parameterObj = new IDS_Facet_Parameter(
16                 parameterName.replace("ids:", ""),
17                 parameterRestriction.replace("ids:", ""),
18                 parameterValue);
19             parameters.push(parameterObj);
20         }
21     }
22     return facets;
23 };

```

Figure 32: Method to Obtain the Facet and Parameters of the Applicability or Requirement

In Figure 33, it is disposed of the “queryByEntity” function. In the IFC model, every entity has an "IFC Class" Name and may have a predefined type, which are the two parameters for this facet. In this function, only the Name parameter is being covered, and it is obtaining all the IDs that have a specific “IFC Class” through the iteration on a list of express IDs of the spatial structure obtained in the process represented in Figure 26.

```
1  async function queryByEntity(entityParameters){
2    for(const parameter of entityParameters){
3      if(parameter.Restriction === "simpleValue"){
4        if(parameter.ParameterName === "name"){
5          const filteredArray = flattenedExpressIds.filter(
6            obj => obj.type.includes(parameter.ParameterValue));
7          const arrayOfIds = filteredArray.map(obj => obj.expressID);
8          return arrayOfIds;
9        }
10     }
11  }
12  };
```

Figure 33: Function to Query the Elements by the Entity Facet

The next function implemented is the “*queryByAttributes*”. In the IFC Schema, every entity has a limited set of fundamental data which is called Attribute. For this facet, it is implemented the parameters name and value, and these data can be accessed with IFC.js using the method “*getItemProperties*”, which will retrieve this fundamental set of data. In the implemented function, the IFC.js method is called for each id of the spatial structure, and it is checked against the parameter’s values, to obtain the representative subset. The function is presented in Figure 34.

```
1  async function queryByAttributes(attributesParameters){
2    const arrayOfIds = [];
3    for(const parameter of attributesParameters){
4      if(parameter.Restriction === "simpleValue"){
5        if(parameter.ParameterName === "Name"){
6          for(const id of flattenedExpressIds){
7            const materialProps = await viewer.IFC.loader.ifcManager
8              .getItemProperties(0, id);
9            const filteredArray = materialProps.filter(
10              obj => obj.Name.includes(parameter.ParameterValue));
11            if(filteredArray.length>0){arrayOfIds.push(id);}
12          }
13        }if(parameter.ParameterName === "Value"){
14          for(const id of flattenedExpressIds){
15            const materialProps = await viewer.IFC.loader.ifcManager
16              .getItemProperties(0, id);
17            const filteredArray = materialProps.filter(
18              obj => obj.Value.includes(parameter.ParameterValue));
19            if(filteredArray.length>0){arrayOfIds.push(id);}
20          }
21        }
22      }
23    }
24    return arrayOfIds;
25  };
```

Figure 34: Method to Query the Elements by the Attributes Facet



The query for the Material facet is represented in Figure 35, and it covers only the condition of having the parameter Value. It works similarly to the function for the Attributes facet. It runs the IFC.js method *getMaterialProperties()* across the whole spatial structure searching for matches against the parameter value.

```
1  async function queryByMaterial(materialParameters){
2    const arrayOfIds = [];
3    for(const parameter of materialParameters){
4      if(parameter.Restriction === "simpleValue"){
5        if(parameter.ParameterName === "Value"){
6          for(const id of flattenedExpressIds){
7            const materialProps = await viewer.IFC.loader.ifcManager
8              .getMaterialsPropertiess(0, id);
9            const filteredArray = materialProps.filter(
10              obj => obj.Name.includes(parameter.ParameterValue));
11            if(filteredArray.length>0){arrayOfIds.push(id);}
12          }
13        }
14      }
15    }
16    return arrayOfIds;
17  };
```

Figure 35: Method to Query the Elements by the Materials

The facet with the biggest number of parameters is the Properties. It can be covered by *PropertySet*, Name, Value, Data Type, and URI as parameters. In this application, only three were covered: *PropertySet*, which is responsible for checking if an entity has a specific *PropertySet*; Name, which checks if the entity has a property with a specific name, and Value which evaluates the actual value of a property. To implement this again was used a very similar process, going to the web-ifc-three layer of IFC.js to use the *propertySetParameters()* method, and running it through the IDs of the IFC elements.

Therefore, using this method, it is possible to check if the elements have a specific property set by checking the names in the array of property sets returned by the method. Then, iterating through the property set is possible to check if there is a property with a specific name and/or with a specific value. In this form, it covers these three types of parameters. In Figure 36, the implementation is demonstrated.

```
1  async function queryByPropertySet(propertySetParameters){
2      const arrayOfIds = [];
3      for(const parameter of propertySetParameters){
4          if(parameter.Restriction === "simpleValue"){
5              if(parameter.ParameterName === "PropertySet"){
6                  for(const id of flattenedExpressIds){
7                      const propertySets = await viewer.IFC.loader.ifcManager
8                          .propertySetParameters(0, id);
9                      const filteredArray = propertySets.filter(
10                         obj => obj.Name.includes(parameter.ParameterValue));
11                     if(filteredArray.length>0){arrayOfIds.push(id);}
12                 }
13             }
14             if(parameter.ParameterName === "Name"){
15                 for(const id of flattenedExpressIds){
16                     const propertySets = await viewer.IFC.loader
17                         .propertySetParameters(0, id);
18                     for(set of propertySets)
19                         {
20                             const filteredArray = set.HasProperties.filter(
21                                 obj => obj.Name.includes(parameter.ParameterValue));
22                             if(filteredArray.length>0){arrayOfIds.push(id);}
23                         }
24                 }
25             }
26             if(parameter.ParameterName === "Value"){
27                 for(const id of flattenedExpressIds){
28                     const propertySets = await viewer.IFC.loader
29                         .propertySetParameters(0, id);
30                     for(set of propertySets)
31                         {
32                             const filteredArray = set.HasProperties.filter(
33                                 obj => obj.NominalValue.includes(parameter.ParameterValue)
34                             );
35                             if(filteredArray.length>0){arrayOfIds.push(id);}
36                         }
37                 }
38             }
39         }
40     return arrayOfIds;
41 };
```

Figure 36: Method to Query the Elements by the Properties Facet

The process to evaluate the elements with the Classification and PartOf facets is considerably different compared to the others since first is needed to obtain the correct entity relationships. For the Classification facet, represented in Figure 37, all the individuals of the *IfcRelAssociatesClassification* are selected with the IFC.js method *getAllItemsOfType()*. The objects of this entity store the Relating Classification and the Related Objects. In this way, it covers the parameters: System and Value. Since the relating classification can be the Classification System itself (e.g. UNICLASS 2015) or the reference code inside the classification. Therefore, the System parameter is evaluated by the name of the classification and the Value is managed when the relating classification selected is the reference.

```

1  async function queryByClassification(classificationParameters){
2    const arrayOfIds = [];
3    const classificationRels = await viewer.IFC.loader.ifcManager.ifcManager
4    .getAllItemsOfType(0, IFCREASSOCIATESCLASSIFICATION, true);
5    for(const parameter of classificationParameters){
6      if(parameter.Restriction === "simpleValue"){
7        if(parameter.ParameterName === "System"){
8          for(rel of classificationRels){
9            if(rel.RelatingClassification.Name===parameter.ParameterValue){
10             arrayOfIds.push(rel.RelatedObjects.map(
11               obj => obj.Id));
12           }
13         }
14         const filteredArray = flattenedExpressIds.filter(
15           obj => obj.type.includes());
16         const arrayOfIds = filteredArray.map(obj => obj.expressID);
17       }
18       if(parameter.ParameterName === "Value"){
19         for(rel of classificationRels){
20           if(rel.RelatingClassification.Identification===parameter.ParameterValue){
21             arrayOfIds.push(rel.RelatedObjects.map(
22               obj => obj.Id));
23           }
24         }
25         const filteredArray = flattenedExpressIds.filter(
26           obj => obj.type.includes());
27         const arrayOfIds = filteredArray.map(obj => obj.expressID);
28       }
29     }
30   }
31   return arrayOfIds;
32 };

```

Figure 37: Method to Query the Elements by the Classification Facet

The last query function is for the “part of” facet, which is the most complex, and is shown in Figure 38.

```

1  async function queryByPartOf(partOfParameters){
2    const arrayOfIds = [];
3    for(const parameter of partOfParameters){
4      if(parameter.Restriction === "simpleValue"){
5        if(parameter.ParameterName === "Entity"){
6          const relations = await viewer.IFC.loader.ifcManager
7          .getAllItemsOfType(0, parameter.Relation, true);
8          for(rel of relations){
9            const relatingObj = rel.RelatingObject;
10           const entityType = await viewer.IFC.loader.ifcManager
11           .getType(relatingObj);
12           if(entityType===parameter.entity){
13             arrayOfIds.push(rel.RelatedObjects);
14           }
15         }
16       }
17     }
18   }
19   return arrayOfIds;
20 };

```

Figure 38: Method to Query the Elements by the PartOf Facet

The process is partially similar to the previous one, but the types of relationships can vary according to what is defined on the IDS file. Therefore, in this current implementation, it only covered the scenario that defined the relation type in the file and the relating object, the main object. With this pair of information and some iterations, as can be seen in Figure 38, the related objects can be obtained.

The last step to construct the final list of elements is an iteration through the applicability or requirements facet, supported by a switch case statement to define the correct query functions to be used. This process can be visualized in Figure 39.

```
1  const collectionOfIds = [];  
2    for(const facet of applicability) {  
3      switch(facet.FacetType) {  
4        case "entity":  
5          console.log("Obtaining elements with Facet:" + facet.FacetType);  
6          const elementsIdsEntity = await queryByEntity(facet.Parameters);  
7          collectionOfIds.push(elementsIdsEntity);  
8          break;  
9        case "classification":  
10         console.log("Obtaining elements with Facet:" + facet.FacetType);  
11         const elementsIdsClassification = queryByClassification(facet.Parameters);  
12         collectionOfIds.push(elementsIdsClassification);  
13         break;  
14        case "propertySet":  
15         console.log("Obtaining elements with Facet:" + facet.FacetType);  
16         const elementsIdsPropertySet = queryByPropertySet(facet.Parameters);  
17         collectionOfIds.push(elementsIdsPropertySet);  
18         break;  
19        case "material":  
20         console.log("Obtaining elements with Facet:" + facet.FacetType);  
21         const elementsIdsPMaterial = queryByMaterial(facet.Parameters);  
22         collectionOfIds.push(elementsIdsPMaterial);  
23         break;  
24        case "attributes":  
25         console.log("Obtaining elements with Facet:" + facet.FacetType);  
26         const elementsIdsAttributes = queryByAttributes(facet.Parameters);  
27         collectionOfIds.push(elementsIdsAttributes);  
28         break;  
29        case "partOf":  
30         console.log("Obtaining elements with Facet:" + facet.FacetType);  
31         const elementsIdsPartOf = queryByPartOf(facet.Parameters);  
32         collectionOfIds.push(elementsIdsPartOf);  
33         break;  
34      }  
35    }
```

Figure 39: Switch Case Statement to Select the Query Methods Based on the IDS parsed.

In Figure 40, it is highlighted the frontend of the implementation. The viewer on the left side shows the complete model, and the viewer on the right side presents the subset of data defined by the applicability and colors the elements with green or red according to the analysis of the requirements. The IDS file used in this example is partially shown in Figure 12.

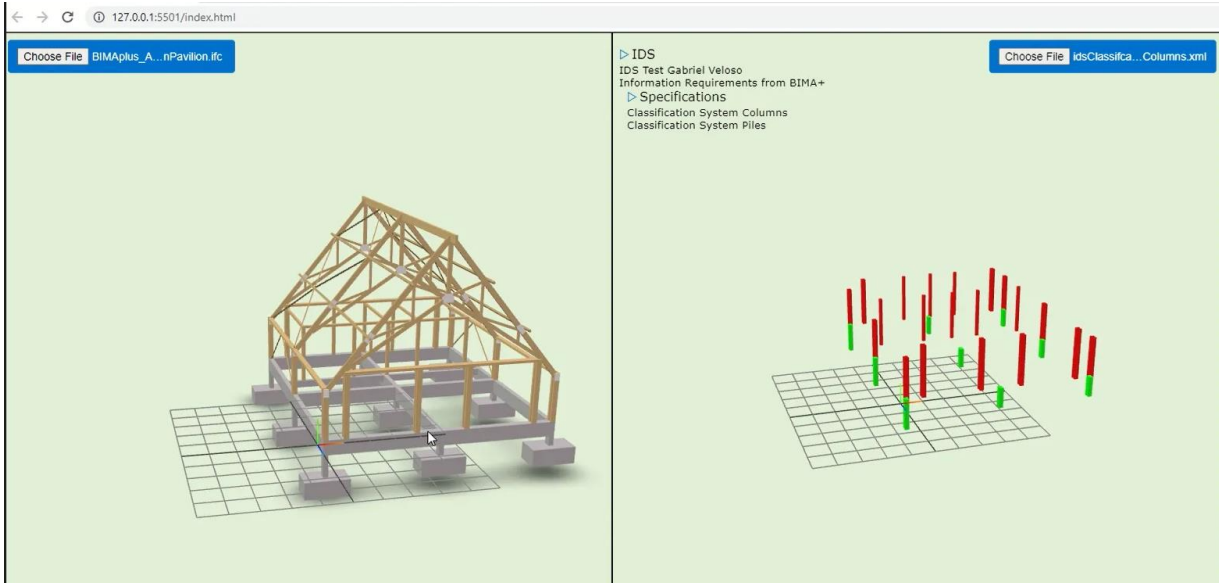


Figure 40: Visualization of the Implementation

In Figure 41, it is presented the console logs of some steps of the test execution that are visualized above. The first line shows some metadata of the IDS file, name, and description. The third and ninth lines show the two specifications that exist inside the file and were reconstructed as objects from the class *IDS\_Specification*. The seventh and thirteenth lines represent the arrays of IDs that form the subsets of data.

```

1 bundle.js:2 (2) ['IDS Test Gabriel Veloso', 'Informati
  on Requirements from BIMA+']
2 dentro de defineSubset bundle.js:2
3 bundle.js:2 M {Occur: 'minOccurs', Name: 'Classificati
  on System Columns', Applicability: Array
  (1), Requirements: Array(1)}
4 > [x] bundle.js:2
5 Obtaining elements with bundle.js:2 Facet:entity
6 Criando subset bundle.js:2
7 bundle.js:2 (32) [190, 232, 262, 282, 1420, 1440, 146
  0, 1480, 1500, 1520, 1540, 1560, 314, 356,
  386, 406, 796, 826, 846, 866, 886, 1212, 1
  232, 1252, 1272, 1580, 1600, 1620, 1738, 1
  758, 1898, 1918]
8 dentro de defineSubset bundle.js:2
9 bundle.js:2 M {Occur: 'minOccurs', Name: 'Classificati
  on System Piles', Applicability: Array(1),
  Requirements: Array(1)}
10 > [x] bundle.js:2
11 Obtaining elements with bundle.js:2 Facet:entity
12 Criando subset bundle.js:2
13 bundle.js:2 (9) [3255, 3353, 3450, 3519, 3588, 3657, 3
  726, 3795, 3864]

```

Figure 41: Console logs of the test


### 4.3 Integration with Blockchain, Smart Contracts, and NFT Collections

As mentioned in section 4.1, the backend of the Decentralized Application for the Marketplace is built with the Motoko Language. This programming language is designed to run distributed applications on the Internet Computer blockchain network (Russo, 2022). The code of this DApp was developed on top of the application developed by Yu, A(2022), which can be used under the Apache License 2.0. Before

diving into the code structure of the backend, it is important to define three key concepts within this network:

- **Canisters:** Canister smart contracts, or canisters, are smart contracts on the Internet Computer that comprise a bundle of Web Assembly (Wasm) bytecode and smart contract data storage. Each canister has its own, separate data storage that is only modified when the code in the canister is executed (Internet Computer, 2022.).
- **Actors:** A Motoko actor represents a canister smart contract. An actor is a self-contained object that communicates with other actors solely through asynchronous messages (Russo, 2022). Actor classes in Motoko enable you to create networks of actors programmatically.
- **Principal:** is an opaque Type that represents canisters and users and can be used to identify callers of shared functions and for basic authentication. Although principals are opaque, they can be transformed into binary Blob values for more efficient hashing and other purposes.

For the backend of this DApp, two canisters were defined to perform interactions between the functions off and on chain. The first is the NFT itself, which is a type of Smart Contract by definition. This canister is structured to receive the metadata that forms the NFTs and, to provide the services to share information about the NFT and to transfer the ownership during the tender process. Figure 42 presents the candid file of the NFT Canister, which is built along the deployment of the canister.



```
1 type NFT =
2   service {
3     getAsset: () -> (vec nat8) query;
4     getCanisterId: () -> (principal) query;
5     getSpecificationName: () -> (text) query;
6     getOwner: () -> (principal) query;
7     transferOwnership: (principal) -> (text);
8   };
9   service : (text, principal, vec nat8) -> NFT
```

Figure 42: NFT Canister Candid File

The second canister was called *OpenBIM\_D* and has the responsibility to provide management services for the processes that are running on the Marketplace. Figure 42 shows the services provided by this canister. These services are structured to cover the processes highlighted in the user stories following the ISO 19650 stages. As an example, the *mint* function would be used during stage 5.1, the *listItem* function in 5.2 to make available the services in the Marketplace, the *resquestConfirmation* during the 5.3 as part of the Tender response process, the *completeTransfer* in the 5.4 as a confirmation of the Appointment by transferring the ownership of the NFT, the *requestApproval* during the 5.6 to order the approval of the service by the Lead Appointed Party or Appointing Party, and the *givingApproval* in stages 5.7 and 5.8 to confirm the information delivery and give reputation to the execution. The functions

which start with “is” or “get” are responsible for checking the state of some processes or obtaining information from the canisters and reporting it to the frontend.

```

1  service : {
2    completeTransfer: (principal, principal, principal) -> (text);
3    getListedNFTPrice: (principal) -> (nat) query;
4    getListedNFTs: () -> (vec principal) query;
5    getOpenBIMDCanisterID: () -> (principal) query;
6    getOriginalOwner: (principal) -> (principal) query;
7    getOwnedNFTs: (principal) -> (vec principal) query;
8    givingApproval: (principal, nat, nat) -> (text);
9    isApprovalGiving: (principal) -> (bool) query;
10   isApprovalRequested: (principal) -> (bool) query;
11   isListed: (principal) -> (bool) query;
12   isWaitingConfirmation: (principal) -> (bool) query;
13   listItem: (principal, nat) -> (text);
14   mint: (vec nat8, text) -> (principal);
15   requestApproval: (principal, nat) -> (text);
16   requestConfirmation: (principal, nat) -> (text);
17 }

```

Figure 43: *OpenBIM\_D* Canister Candid File

As it is possible to see in both candid files, some of the functions are tagged with **query**. This type of function solely can get information from the canisters. As an example, in Figure 44 is shown the process of getting all the owned NFTs and returning a list of identifications (Principal) of the owners.

```

1
2  public query func getOwnedNFTs(user: Principal) : async [Principal] {
3    var userNFTs : List.List<Principal> = switch (mapOfOwners.get(user)) {
4      case null List.nil<Principal>();
5      case (?result) result;
6    };
7
8    return List.toArray(userNFTs);
9  };

```

Figure 44: Query Function Example

Another type of function is **shared**, which identifies the callers by the **Principal** type. Differently from the **Query** functions, this type executes a process as minting an NFT or transferring the ownership of it. Therefore, the Shared functions are accessible to remote callers, and they are the only way to modify the state of an Actor.

In Figure 45, the function to mint the NFTs is presented as an example of a shared function. In this function, the caller is identified as the initial owner of the NFT, and then, in the process of minting, the Principal of the owner is passed as an argument together with the bytes data of the IDS file and its name.

```
1 public shared(msg) func mint(fileData: [Nat8], name: Text) : async Principal {
2   let owner : Principal = msg.caller;
3
4   Debug.print(debug_show(Cycles.balance()));
5   Cycles.add(100_500_000_000);
6   let newNFT = await NFTActorClass.NFT(name, owner, fileData);
7   Debug.print(debug_show(Cycles.balance()));
8
9   let newNFTPrincipal = await newNFT.getCanisterId();
10
11  mapOfNFTs.put(newNFTPrincipal, newNFT);
12  addToOwnershipMap(owner, newNFTPrincipal);
13
14  return newNFTPrincipal
15
16  };
```

Figure 45: Function Mint

Figure 46 presents another important function to create Actors. The actors are created to represent the canister, in this case, the NFTs that were minted.

```
1 async function LoadNFT() {
2   NFTActor = await Actor.createActor(idlFactory, {
3     agent,
4     canisterId: id,
5   });
6 }
```

Figure 46: Creating Actors

With the process to mint the construction services as NFTs and functions to operate the marketplace implemented, the next step was to define the mechanism to integrate this workflow with the IFC entities of the models that are the reference for the execution of the construction services. In Figure 47, it is shown how this integration was designed in this application.



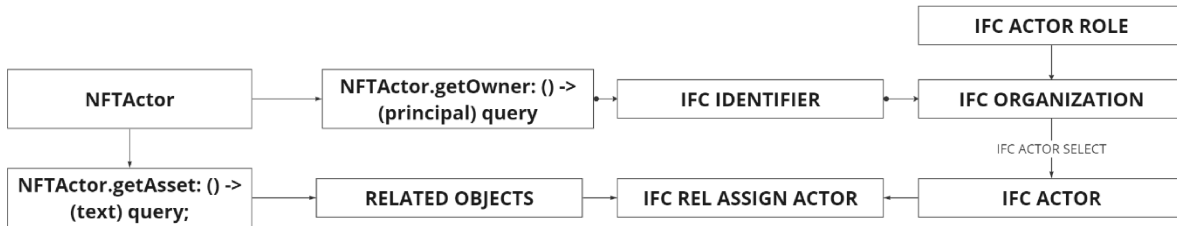


Figure 47: Integration between NFT Metadata and IFC Entities

This integration is based on storing the Owner Principal in an IFC Identifier, which can be used to define an *IfcOrganization*. Since, the NFTs reflect construction services, and the final owners are the Lead Appointed Parties for this phase, it is added to the organization the role of the Contractor through the IFC Actor Role entity. Then, the organization is selected to be the Actor that acts upon the IFC Objects that the Delivery Specification stored in the NFT defines.

The process to write new information inside of the IFC files was managed in the frontend using the IFC.js library. The library has no layer to simply the writing process, it is required to use the *WriteLine()* function from the Web-IFC combined with the exact structure of the IFC Entity that is wanted to be written. Therefore, it was needed to create a specialized function for each IFC entity, and their EXPRESS Specification supported this process, especially to obtain the mandatory Attributes. In Figure 48 it is shown the *IfcOrganization* EXPRESS specification, which defines the Name as the only mandatory attribute, and points to the Identification as an *IfcIdentifier*, being the best container for the owner Principal data, since it is qualified as an Identification.

### EXPRESS Specification

```

ENTITY IfcOrganization;
  Identification : OPTIONAL IfcIdentifier;
  Name : IfcLabel;
  Description : OPTIONAL IfcText;
  Roles : OPTIONAL LIST [1:?] OF IfcActorRole;
  Addresses : OPTIONAL LIST [1:?] OF IfcAddress;
INVERSE
  IsRelatedBy : SET [0:?] OF IfcOrganizationRelationship FOR RelatedOrganizations;
  Relates : SET [0:?] OF IfcOrganizationRelationship FOR RelatingOrganization;
  Engages : SET [0:?] OF IfcPersonAndOrganization FOR TheOrganization;
END_ENTITY;

```

[EXPRESS-G diagram](#)

Figure 48: *IfcOrganization* EXPRESS Specification (buildingSMART International, 2019)

In the implemented function, highlighted in Figure 49, it is generated the *IfcIdentifier* with the Hash, an *IfcLabel* with the name of the company, an *IfcText* with the company description, and a *IfcActorRole* entity, to construct the *IfcOrganization* object.

```

1  export async function createIfcOrganization(manager, modelID, companyName, companyDescription, roleDescription, identifierHash){
2
3      const ifcIdOrg = await manager.ifcAPI.CreateIfcType(modelID, WebIfc.IFCIDENTIFIER, identifierHash);
4      const ifcLabelCompanyName = await manager.ifcAPI.CreateIfcType(modelID, WebIfc.IFCLABEL, companyName);
5      const ifcTextCompanyDescription = await manager.ifcAPI.CreateIfcType(modelID, WebIfc.IFCTEXT, companyDescription);
6      const args = ifcIdOrg;
7      const ifcEntityOrg = await manager.ifcAPI.CreateIfcEntity(modelID, WebIfc.IFCORGANIZATION, args);
8      ifcEntityOrg.Name = ifcLabelCompanyName;
9      ifcEntityOrg.Description = ifcTextCompanyDescription;
10     const ifcEntityRole = await createIfcActorRole(manager, modelID, roleDescription);
11     const roles = [ifcEntityRole];
12     ifcEntityOrg.Roles = roles;
13     manager.ifcAPI.WriteLine(modelID, ifcEntityOrg);
14     console.log(ifcEntityOrg);
15     return ifcEntityOrg;
16 }

```

Figure 49: Function to create IfcOrganization Entity

The EXPRESS Specification of the *IfcActorRole* entity presented in Figure 50, highlights the Role as the only mandatory attribute receiving an Enum to define which exact role that actor will play.

### EXPRESS Specification

```

ENTITY IfcActorRole;
  Role : IfcRoleEnum;
  UserDefinedRole : OPTIONAL IfcLabel;
  Description : OPTIONAL IfcText;
INVERSE
  HasExternalReference : SET [0:?] OF IfcExternalReferenceRelationship FOR RelatedResourceObjects;
WHERE
  WR1 : (Role <> IfcRoleEnum.USERDEFINED) OR
  ((Role = IfcRoleEnum.USERDEFINED) AND
  EXISTS(SELF.UserDefinedRole));
END_ENTITY;

```

Figure 50: *IfcActorRole* EXPRESS Specification (buildingSMART International, 2019)

In Figure 51 it is displayed how is defined the function to write the *IfcActorRole*, which is called in the function to create the *IfcOrganization*. At this point, is established the role as a Contractor.

```

1  async function createIfcActorRole(manager, modelID, roleDescription){
2
3      const ifcRoleEnum = await manager.ifcAPI.CreateIfcType(modelID, WebIfc.IFCINTEGER, IFC4.IfRoleEnum.CONTRACTOR);
4      const ifcEntityRole = await manager.ifcAPI.CreateIfcEntity(modelID, WebIfc.IFCACTORROLE, ifcRoleEnum);
5      ifcEntityRole.Description = await manager.ifcAPI.CreateIfcType(modelID, WebIfc.IFCTEXT, roleDescription);
6      manager.ifcAPI.WriteLine(modelID, ifcEntityRole);
7      console.log(ifcEntityRole);
8      return ifcEntityRole;
9  }

```

Figure 51: Function to create *IfcActorRole* Entity

In Figure 52 it is presented the output of the function *createIfcOrganization()* in a console log. It is possible to observe the Identification being an *IfcIdentifier* which the value would come from the NFT

metadata. Another relevant point to observe is the Attribute Roles receiving the Array of *IfcActorRole* as it was defined.

```

bundle.js:139533
▼ IfcOrganization {expressID: 557, type: 4251960020, Identification: IfcIdentifier, Name: IfcLabel, D
  escription: IfcText, ...} ⓘ
  Addresses: null
  ▶ Description: IfcText {value: 'companycheck', type: 1}
  ▶ Identification: IfcIdentifier {value: 'dfds0#@ds', type: 1}
  ▶ Name: IfcLabel {value: 'companyBIMA', type: 1}
  ▼ Roles: Array(1)
    ▶ 0: IfcActorRole {expressID: 556, type: 3630933823, Role: {...}, UserDefinedRole: IfcLabel, Descrip
      length: 1
    ▶ [[Prototype]]: Array(0)
    expressID: 557
    type: 4251960020
  ▶ [[Prototype]]: IfcLineObject

```

Figure 52: Console log of *IfcOrganization* with Role

Therefore, the next step is to generate an actual *IfcActor* selecting the organization as *TheActor*, which is defined in Figure 53.

### EXPRESS Specification

```

ENTITY IfcActor
  SUPERTYPE OF (IfcOccupant)
  SUBTYPE OF (IfcObject);
  TheActor : IfcActorSelect;
  INVERSE
  IsActingUpon : SET [0:?] OF IfcRelAssignsToActor FOR RelatingActor;
END_ENTITY;
EXPRESS-G diagram

```

Figure 53: *IfcActor* EXPRESS Specification (buildingSMART International, 2019)

All the previous entities are from the Resource Definition layer of the IFC Schema, which differs from the *IfcActor*, which is defined in the Core Data Schema inheriting from the *IfcKernel*. Thus, in the function to create the *IfcActor*, presented in Figure 54, it was necessary to generate a Global Unique ID and an Owner History, which both could also be considered to connect with the NFT Metadata.

```

1 export async function createIfcActor(manager, modelID, organization){
2
3   const ifcIdActor = await manager.ifcAPI.CreateIfcType(modelID, WebIfc.IFCGLOBALLYUNIQUEID, generateGlobalUniqueId());
4   const ifcEntityActor = await manager.ifcAPI.CreateIfcEntity(modelID, WebIfc.IFCACTOR, ifcIdActor);
5   ifcEntityActor.OwnerHistory = await createIfcOwnerHistory(manager, modelID, 2023);
6   manager.ifcAPI.WriteLine(modelID, ifcEntityActor);
7   ifcEntityActor.TheActor = organization;
8   console.log(ifcEntityActor);
9   return ifcEntityActor;
10 }

```

Figure 54: Function to Create *IfcActor* entity (buildingSMART International, 2019)

In Figure 55, it is presented the console log of the return of this function, demonstrating that *IfcOrganization* is correctly assigned as *TheActor*, which can be evaluated by the value that is the express ID generated for the entity in Figure 52.

```

bundle.js:62024
▼ IfcActor {expressID: 559, type: 2296667514, GlobalId: IfcGloballyUniqueId, OwnerHistory: Handle, Na
me: undefined, ...} ⓘ
  Description: undefined
  ▶ GlobalId: IfcGloballyUniqueId {value: '31287406255', type: 1}
  Name: undefined
  ObjectType: undefined
  ▶ OwnerHistory: Handle {value: 558, type: 5}
  ▼ TheActor: Handle
    type: 5
    value: 557
    ▶ [[Prototype]]: Object
    expressID: 559
    type: 2296667514
    ▶ [[Prototype]]: IfcObject

```

Figure 55: Console log of *IfcActor* Entity (buildingSMART International, 2019)

The last step of this implementation is to create the relationship between the actor to what he is acting upon. This process is handled by the *IfcRelAssignsToActor* entity, which EXPRESS Specification is represented in Figure 56.

#### EXPRESS Specification

```

ENTITY IfcRelAssignsToActor
SUBTYPE OF (IfcRelAssigns);
RelatingActor : IfcActor;
ActingRole : OPTIONAL IfcActorRole;
WHERE
NoSelfReference : SIZEOF(QUERY(Temp <* SELF\IfcRelAssigns.RelatedObjects | RelatingActor := Temp)) = 0;
END_ENTITY;

```

Figure 56: *IfcRelAssignsToActor* EXPRESS Specification (buildingSMART International, 2019)

Therefore, in Figure 57, it is demonstrated the function to create a relationship entity, which connects the Actor, represented by the Organization, the owner of the construction service NFT, with the related objects defined by the specification stored in the NFT.

```

1 export async function createIfcRelAssignsToActor(manager, modelID, ifcEntityActor, relatedObjects){
2   const ifcIdRelAssignActor = await manager.ifcAPI.CreateIfcType(modelID, WebIfc.IFCGLOBALLYUNIQUEID, generateGlobalUniqueId());
3   const ifcEntityRelAssignActor = await manager.ifcAPI.CreateIfcEntity(modelID, WebIfc.IFCRELAASSIGNSTOACTOR, ifcIdRelAssignActor);
4   ifcEntityRelAssignActor.RelatingActor = ifcEntityActor;
5   ifcEntityRelAssignActor.RelatedObjectsType = IFC4.IfcObjectTypeEnum.PRODUCT;
6   ifcEntityRelAssignActor.RelatedObjects = relatedObjects;
7   manager.ifcAPI.WriteLine(modelID, ifcEntityRelAssignActor);
8   console.log(ifcEntityRelAssignActor);
9   return ifcEntityRelAssignActor;
10 }

```

Figure 57: Function to create *IfcRelAssignsToActor* Entity

#### 4.4 Frontend of the Decentralized Marketplace

The frontend of Decentralized Marketplace for Construction Services is also adapted from Yu, A application. It is constructed by several React Components that are dynamically reused to form the four main pages of the DApp: **Marketplace**, **Minter**, **My NFTs**, and **My Models**. In Figure 58, it is illustrated the components outline and their relationship.

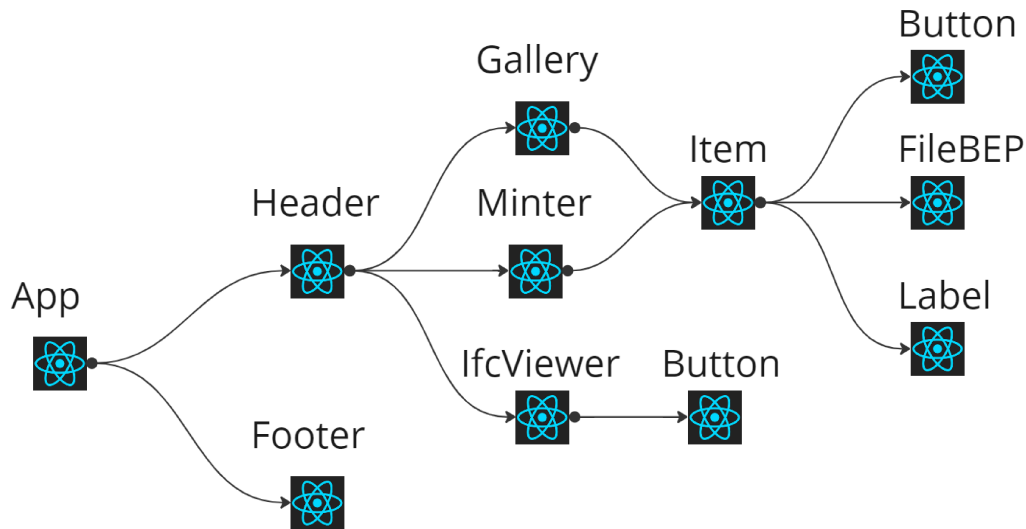


Figure 58: React Components Diagram

Each component has a responsibility, which is described above:

- **App**: this component has the responsibility to instantiate the Header and Footer components.
- **Footer**: as the name suggests, it is a component to compose the bottom of the web pages.
- **Header**: the header is a key component for the frontend, it is responsible to output the main content for each page. For “My NFTs” and “Marketplace”, it will display the Gallery component, for the Minter page the Minter, and for the “My Models” the *IfcViewer*.
- **Gallery**: component to display the collection of NFTs as Items, the NFTs available for the tender will be displayed in the Marketplace, and the ones with defined owners, in “My NFTs”.
- **Minter**: has the function to display the forms to get the NFT data, and then call the function to Mint it.
- **IfcViewer**: component to instantiate the IFC viewer.
- **Item**: this component has the responsibility to present an NFT, obtaining all the required data from the canisters to display in the Frontend.
- **Button**: component to display buttons to handle events to call different functions.
- **FileBEP**: component displays a specific input to upload BEP files.
- **Label**: it is a component to generate labels for the Item.

Some of the components directly interact with the blockchain canisters, through specific functions. These relationships are described in **Table 6**.

**Table 6:** React Components Interactions with Canisters

| React Component | Interacts With        | By the function                |
|-----------------|-----------------------|--------------------------------|
| Item            | <i>NFTActor</i>       | <i>getSpecificationName()</i>  |
| Item            | <i>NFTActor</i>       | <i>getOwner()</i>              |
| Item            | <i>NFTActor</i>       | <i>getAsset()</i>              |
| Item            | <i>OpenBIM_DActor</i> | <i>isListed()</i>              |
| Item            | <i>OpenBIM_DActor</i> | <i>isWaitingConfirmation()</i> |
| Item            | <i>OpenBIM_DActor</i> | <i>getOriginalOwner()</i>      |
| Item            | <i>OpenBIM_DActor</i> | <i>getListedNFTPrice()</i>     |
| Item            | <i>OpenBIM_DActor</i> | <i>listItem()</i>              |
| Item            | <i>OpenBIM_DActor</i> | <i>getOpenBIMDCanisterID()</i> |
| Item            | <i>NFTActor</i>       | <i>transferOwnership()</i>     |
| Item            | <i>OpenBIM_DActor</i> | <i>completeTransfer()</i>      |
| Minter          | <i>OpenBIM_DActor</i> | <i>mint()</i>                  |
| IfcViewer       | <i>OpenBIM_DActor</i> | <i>requestApproval()</i>       |
| IfcViewer       | <i>OpenBIM_DActor</i> | <i>givingApproval()</i>        |
| IfcViewer       | <i>OpenBIM_DActor</i> | <i>isApprovalRequested()</i>   |
| IfcViewer       | <i>OpenBIM_DActor</i> | <i>isApprovalGiving()</i>      |

The output of the full application will be presented in section 4.6 step by step, giving the user's point of view.

#### 4.5 Integration of DID Documents

The objective of integrating DID Documents in this application is to mainly focus on the attachment of proofs of the execution of the services on the construction site. The designed process to perform this integration is presented in Figure 59.

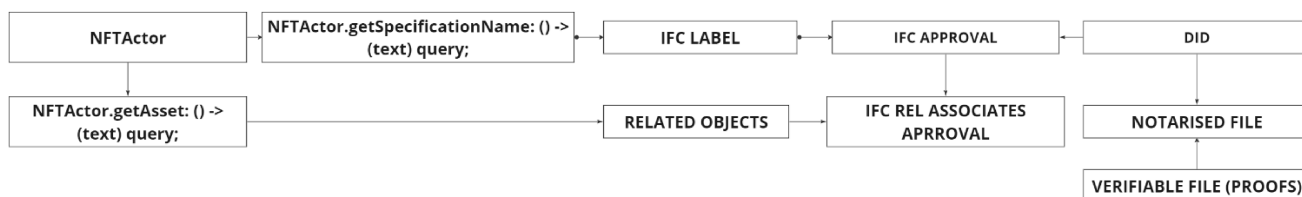


Figure 59: Integration of DID Documents

The key to this process is to create an *IfcApproval* entity to be the container of information related to the execution of the service. In a similar way to the *IfcActor* integration with the NFT Metadata, here it is also used the NFT Asset to define the Related Objects to this approval process and the Specification Name labels the entity. In Figure 60, it is shown the *IfcApproval* EXPRESS Specification, which provides the attributes to allocate the Actor who is requesting the Approval, the owner of the NFT, and the Actor who will give the Approval, the original owner of the NFT. It also highlights the *IfcRelAssociatesApproval* entity as the to create the relationship between the approval and the objects. Another relevant point is the *IfcExternalReferenceRelationship* which is used to point to External References and could be a possible fit to make a connection with the DID Documents in order to attach the verifiable proofs of execution.

### EXPRESS Specification

```
ENTITY IfcApproval;
  Identifier : OPTIONAL IfcIdentifier;
  Name : OPTIONAL IfcLabel;
  Description : OPTIONAL IfcText;
  TimeOfApproval : OPTIONAL IfcDateTime;
  Status : OPTIONAL IfcLabel;
  Level : OPTIONAL IfcLabel;
  Qualifier : OPTIONAL IfcText;
  RequestingApproval : OPTIONAL IfcActorSelect;
  GivingApproval : OPTIONAL IfcActorSelect;
INVERSE
  HasExternalReferences : SET [0:?] OF IfcExternalReferenceRelationship FOR RelatedResourceObjects;
  ApprovedObjects : SET [0:?] OF IfcRelAssociatesApproval FOR RelatingApproval;
  ApprovedResources : SET [0:?] OF IfcResourceApprovalRelationship FOR RelatingApproval;
  IsRelatedWith : SET [0:?] OF IfcApprovalRelationship FOR RelatedApprovals;
  Relates : SET [0:?] OF IfcApprovalRelationship FOR RelatingApproval;
WHERE
  HasIdentifierOrName : EXISTS (Identifier) OR EXISTS (Name);
END_ENTITY;
```

Figure 60: *IfcApproval* EXPRESS Specification (buildingSMART International, 2019)

In Figure 61 is presented the function with IFC.js to create the *IfcApproval* entity, establishing the actors that are requesting and giving the approval.

```
1 export async function createIfcApproval(manager, modelID, approvalName, approvalDescription, actorGivinigApproval, actorRequestingApproval)
2 {
3   const args = await manager.ifcAPI.CreateIfcType(modelID, WebIfc.IFCIDENTIFIER, generateGlobalUniqueId());
4   const ifcEntityApproval = await manager.ifcAPI.CreateIfcEntity(modelID, WebIfc.IFCAPPROVAL, args);
5   //ifcEntityApproval.ApprovalDateTime = await manager.ifcAPI.CreateIfcType(modelID, WebIfc.IFCTIMESTAMP, Date.now()); //IFC2X3
6   ifcEntityApproval.Name = await manager.ifcAPI.CreateIfcType(modelID, WebIfc.IFCLABEL, approvalName);
7   ifcEntityApproval.Description = await manager.ifcAPI.CreateIfcType(modelID, WebIfc.IFCTEXT, approvalDescription);
8   ifcEntityApproval.Status = await manager.ifcAPI.CreateIfcType(modelID, WebIfc.IFCLABEL, "Requested");
9   ifcEntityApproval.RequestingApproval = actorRequestingApproval; //IFC4
10  ifcEntityApproval.GivingApproval = actorGivinigApproval; //IFC4
11  ifcEntityApproval.TimeOfApproval = await manager.ifcAPI.CreateIfcType(modelID, WebIfc.IFCDATETIME, new Date(Date.now()).toLocaleString());
12  manager.ifcAPI.WriteLine(modelID, ifcEntityApproval);
13  console.log(ifcEntityApproval);
14  return ifcEntityApproval;
15 }
```

Figure 61: Create *IfcApproval* Entity Function

The output of the function is shown in Figure 62 through a console log, using random values, just to validate the structure of the entity.

```
bundle.js:62049
▼ IfcApproval {expressID: 560, type: 130549933, Identifier: IfcIdentifier, Name: IfcLabel, Descriptio
  n: IfcText, ...} ⓘ
  ▶ Description: IfcText {value: 'descrip', type: 1}
  ▶ GivingApproval: Handle {value: 559, type: 5}
  ▶ Identifier: IfcIdentifier {value: '125366243156', type: 1}
    Level: undefined
  ▶ Name: IfcLabel {value: 'name', type: 1}
    Qualifier: undefined
  ▶ RequestingApproval: Handle {value: 559, type: 5}
  ▶ Status: IfcLabel {value: 'Requested', type: 1}
  ▶ TimeOfApproval: IfcDateTime {value: '9/2/2023, 9:25:12 AM', type: 1}
  expressID: 560
  type: 130549933
  ▶ [[Prototype]]: IfcLineObject
```

Figure 62: Console log of the *IfcApproval*

As it was possible to observe in the EXPRESS Specification of the *IfcApproval*, to establish the relationship with the objects to be approved it is required to create an *IfcRelAssociatesApproval* entity. Then, a function was implemented, passing as main arguments an *IfcApproval* itself and an array of objects, obtained by the specification data. This process is defined in Figure 63.

```
1 export async function createIfcRelAssociatesApproval(manager, modelID, ifcapproval, relatedObjects){
2   const ifcEntityApprovalRel = await manager.ifcAPI.CreateIfcEntity(modelID, WebIfc.IFCRELAASSOCIATESAPPROVAL, ifcapproval);
3   ifcEntityApprovalRel.Approval = ifcapproval;
4   ifcEntityApprovalRel.RelatedObjects = relatedObjects;
5   console.log(ifcEntityApprovalRel);
6   manager.ifcAPI.WriteLine(modelID, ifcEntityApprovalRel);
7   console.log(ifcEntityApprovalRel);
8   return ifcEntityApprovalRel;
9 }
```

Figure 63: Create *IfcRelAssociatesApproval* Entity function.

It was shown a possibility to make references to DIDs inside the EXPRESS Schema of IFC. However, the best fit for this operation is to use Linked Data, considering the structure of DIDs. In this way, it is possible to create a semantic relation between both structures, in which the DID Subject is representing the *IfcApproval*, as the diagram presented in Figure 59 can be transposed to an Ontology.

It is not the scope of this work to create the DIDs, or even, to create a graph database to store the data mentioned. The objective of this part is to highlight possible ways to perform the integration. Therefore, the process used to connect the Linked Data to the IFC Model was demonstrated by Donkers Alexand Yang, (2023).

It generated an RDF file containing the information of the *IfcApproval* entity and its relationships, additionally, which brings the EXPRESS IDs of the related objects in the approval process and points



the *IfcApproval* entity to the DID URL. Then, using the *Comunica* package it makes it possible to query this RDF file and highlight in the model rendered in the frontend, the linked data in the knowledge graph, which can bring the verifiable proofs mechanisms of the DID and, the Reputation given by the client for the execution of the service.

#### 4.6 Using the Application Step by Step

To provide a comprehensive understanding of the Decentralized Application's step-by-step usage, it was chosen to simulate the execution of concrete walls as a representative construction service. In Figure 64, the complete model of a building's first floor can be found, featuring concrete walls and slabs. This model serves as the focal point for our demonstration, offering a real-world reference for the application's functionality.

The ISO19650 stages and associated actors, which served as a model for creating the application's user stories, are the ideal way to break down the process and provide context for each step. As a result, the application's use will be divided into the following subsections.

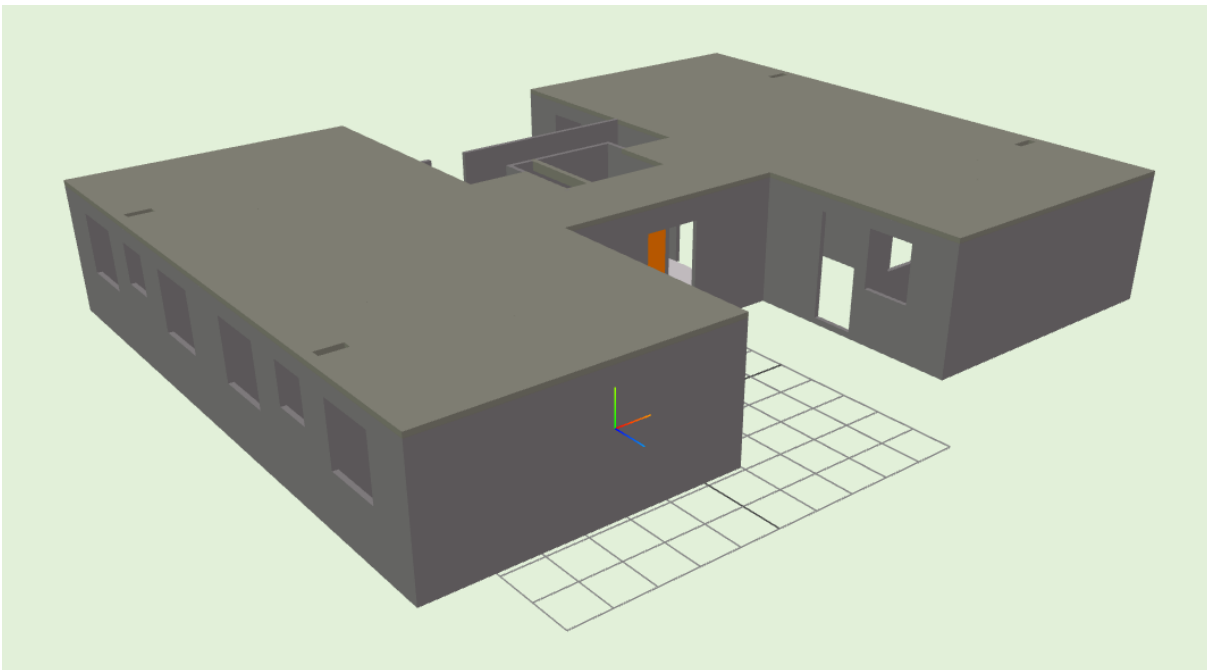


Figure 64: First Floor of the Building

##### 4.6.1 Assessment and Need

The first stage is the assessment and need, which in the application is covered in a very simplistic way, by requesting the Lead Appointing Party to structure the Information Delivery Specification files, that will define the subsets of data and the requirements. Therefore, for this demonstration, a simple IDS was prepared, and it is presented in Figure 65.

This IDS file establishes the Applicability as all the *IfcWall* entities which have the value True for the property named “LoadBearing” in the property set “Pset\_WallCommon”, defining in a machine-readable way, a subset of the model presented in Figure 64. In the Requirements, it is defined that all the elements in the subset shall be given a property named “CompressiveStrengthOnSite” under the “Pset\_WallConcreteMaterial”. Thus, this property will be expected to be used for filling the actual compressive strength obtained after the execution on site.

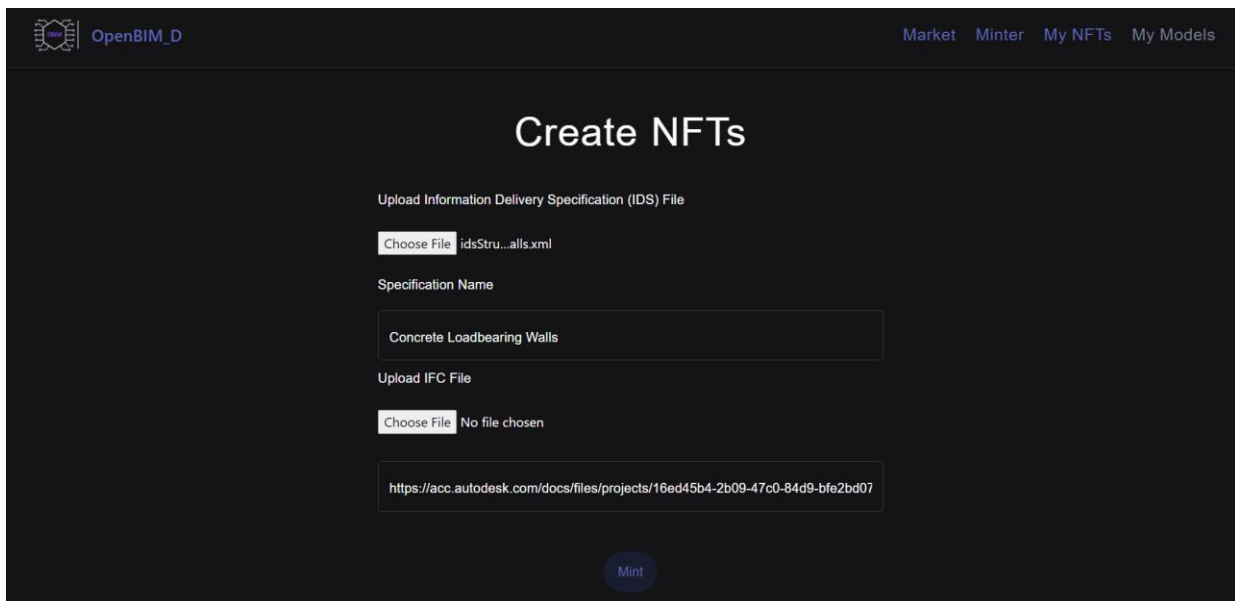
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ids:ids xmlns:ids="http://standards.buildingsmart.org/IDS" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi:schemaLocation="http://standards.buildingsmart.org/IDS/ids_09.xsd">
  <ids:info>
    <ids:title>IDS Master Thesis Gabriel Veloso</ids:title>
    <ids:description>Information Requirements Master Thesis BIMA+</ids:description>
  </ids:info>
  <ids:specifications>
    <ids:specification ifcVersion="IFC4" name="Compressive Strength of Concrete Walls" minOccurs="1">
      <ids:applicability>
        <ids:entity>
          <ids:name>
            <ids:simpleValue>IFCWALL</ids:simpleValue>
          </ids:name>
        </ids:entity>
        <ids:property minOccurs="1" maxOccurs="unbounded" datatype="IFCBOOLEAN">
          <ids:propertySet>
            <ids:simpleValue>Pset_WallCommon</ids:simpleValue>
          </ids:propertySet>
          <ids:name>
            <ids:simpleValue>LoadBearing</ids:simpleValue>
          </ids:name>
          <ids:value>
            <ids:simpleValue>True</ids:simpleValue>
          </ids:value>
        </ids:property>
      </ids:applicability>
      <ids:requirements>
        <ids:property minOccurs="1" maxOccurs="unbounded">
          <ids:propertySet>
            <ids:simpleValue>Pset_WallConcreteMaterial</ids:simpleValue>
          </ids:propertySet>
          <ids:name>
            <ids:simpleValue>CompressiveStrengthOnSite</ids:simpleValue>
          </ids:name>
        </ids:property>
      </ids:requirements>
    </ids:specification>
  </ids:specifications>
</ids:ids>
```

Figure 65: IDS File for Concrete Walls

#### 4.6.2 Invitation to Tender

The process of Invitation to Tender is done by making available in the decentralized marketplace the information about the construction service to the possible Lead Appointed Parties to be able to visualize the data.

Therefore, the first step in the application is to mint the representative NFT, which can be done on the Minter Webpage, shown in Figure 66. There, it will be required to upload the IDS XML file, the name of the specification which will define the construction service, and to create the reference with the IFC Models, which can be done by uploading the IFC file or giving the URL of a CDE folder where the file is stored. Thus, in this demonstration, the IDS from Figure 65 was uploaded, and the specification was named as Concrete Loadbearing Walls.



The screenshot shows the 'Create NFTs' page of the Minter application. The page has a dark background with white text. At the top left, there is a logo for 'OpenBIM\_D' and a navigation menu with 'Market', 'Minter', 'My NFTs', and 'My Models'. The main heading is 'Create NFTs'. Below this, there are three sections: 'Upload Information Delivery Specification (IDS) File' with a 'Choose File' button and the filename 'idsStru...alls.xml'; 'Specification Name' with a text input field containing 'Concrete Loadbearing Walls'; and 'Upload IFC File' with a 'Choose File' button and the text 'No file chosen'. Below these is a text input field containing a URL: 'https://acc.autodesk.com/docs/files/projects/16ed45b4-2b09-47c0-84d9-bfe2bd07'. At the bottom center, there is a blue circular button labeled 'Mint'.

Figure 66: Minter page

After clicking on the button Mint, the application will call the function *mint()* from the *OpenBIM\_D* canister and then will return to the frontend the conclusion of the process, as highlighted in Figure 67



Figure 67: Notification of the Minted NFT

The IFC file in the Mint form was connected by passing the URL of an Autodesk Construction Cloud folder where the file is stored. This integration is possible using the Autodesk Platform Services Data Management API. This option was selected, since the best and most used practice in the AEC Industry is to store BIM files in CDEs, which will handle several workflows, including the versioning of files. With the APIs to connect different systems, it makes it possible to ensure data compilation.

Then, the last step to make available the NFT in the marketplace is to define the value of this service, according to its complexity to be executed. This process can be performed on the My NFTs webpage by the owner of the NFT, the Appointing Party at this moment, as shown in Figure 68, by clicking on the button Send to Tender, which will make appear the Target field, where he can put the value for this work in “BIM Coins”, which would be given to the future Lead Appointed Party as an extra bonification for the job according to the final reputation. When the button Confirm is pressed, the ownership of the NFT is transferred to the Marketplace, and it becomes available in the tender process.

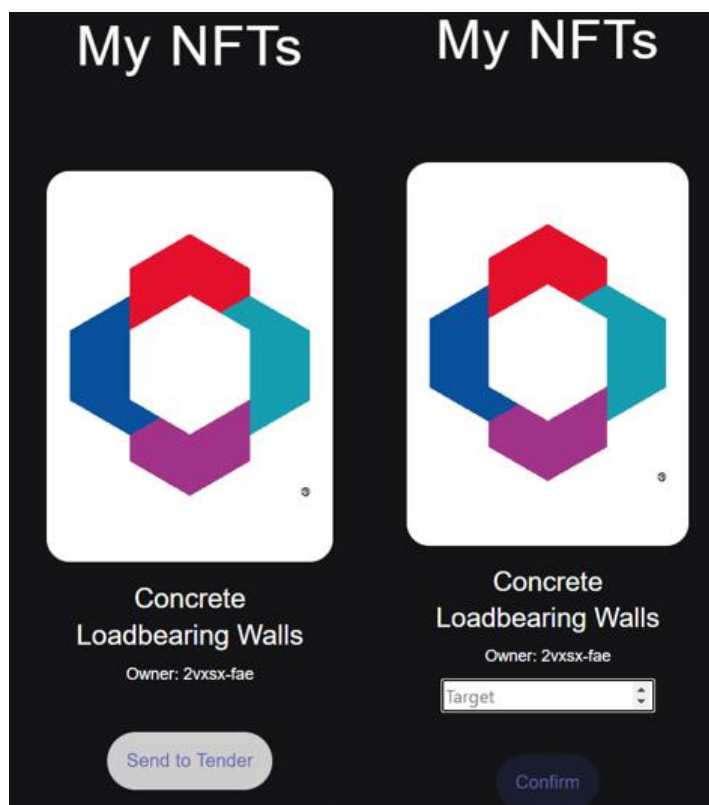


Figure 68: Sending NFTs to Tender

### 4.6.3 Tender Response

The first two stages are done solely by the client, the Appointing Party, it is in the Tender Response that the Lead Appointed Party starts to act. Figure 69 shows how the NFT is presented, after being listed, in the My NFTs webpage for the client (left side), and for the bidders in the Market webpage (right side).

Then, the Lead Appointed Party acts preparing the required documents to respond to the tender, in this simplified application it is requested the pre-appointment BIM Execution Plan. Thus, after uploading the documents, the button response should be pressed.

This button will call the function *requestConfirmation()*, which is a shared function, then, the public identifier of the bidder will be shared with the *OpenBIM\_D* actor. This process is highly important, since, it could be attributed to this public identification, a self-sovereign wallet, which could provide proof of capacity to execute the work.

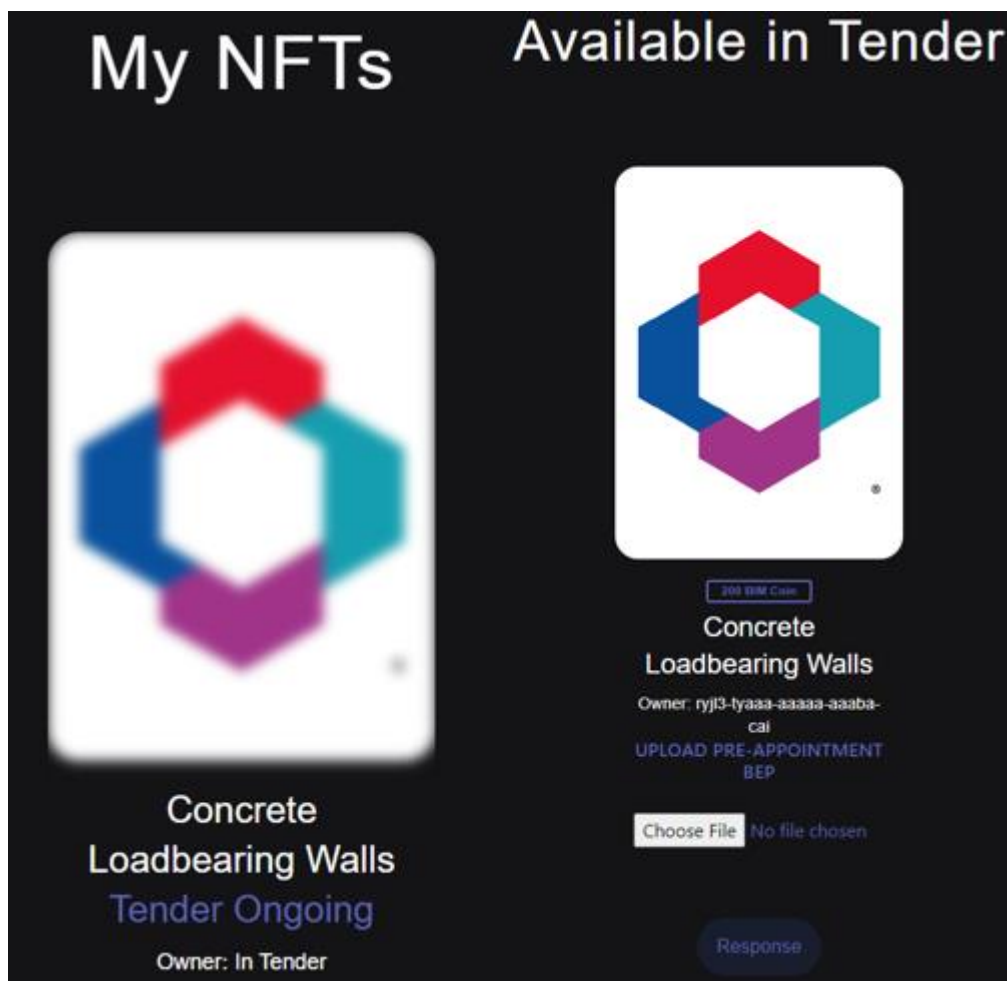


Figure 69: NFT in tender as presented for the Owner and in the Market

#### 4.6.4 Appointment

When clicking on the button response the process is brought back to the Appointing Party, which should evaluate the data and documents received. Then, on the My NFTs webpage will be instantiated the button to Confirm Lead Appointed Party, as can be seen in Figure 70.

By pressing the button, the shared function *completeTransfer()* will be called and the NFT will be finally transferred to the Lead Appointed Party. This process will also trigger the writing of the *IfcOrganization* and *IfcActor* to assign the BIM objects to the contractor, as it was shown in section 4.3.

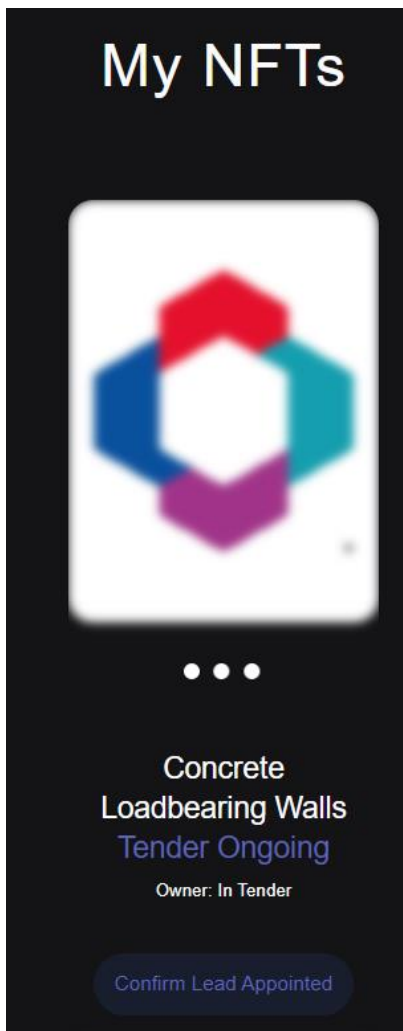


Figure 70: Confirm Lead Appointed Party Button

#### 4.6.5 Mobilization

In the mobilization stage, the application provides the button Update IfcActors, in the My Models webpage, as shown in Figure 71. The button allows the Lead Appointed Party to update *IfcActor* and *IfcActorRole* entities, providing more detail according to what was planned for the execution of the work, especially, if there will be appointed parties, or sub-contractors, involved in the process.

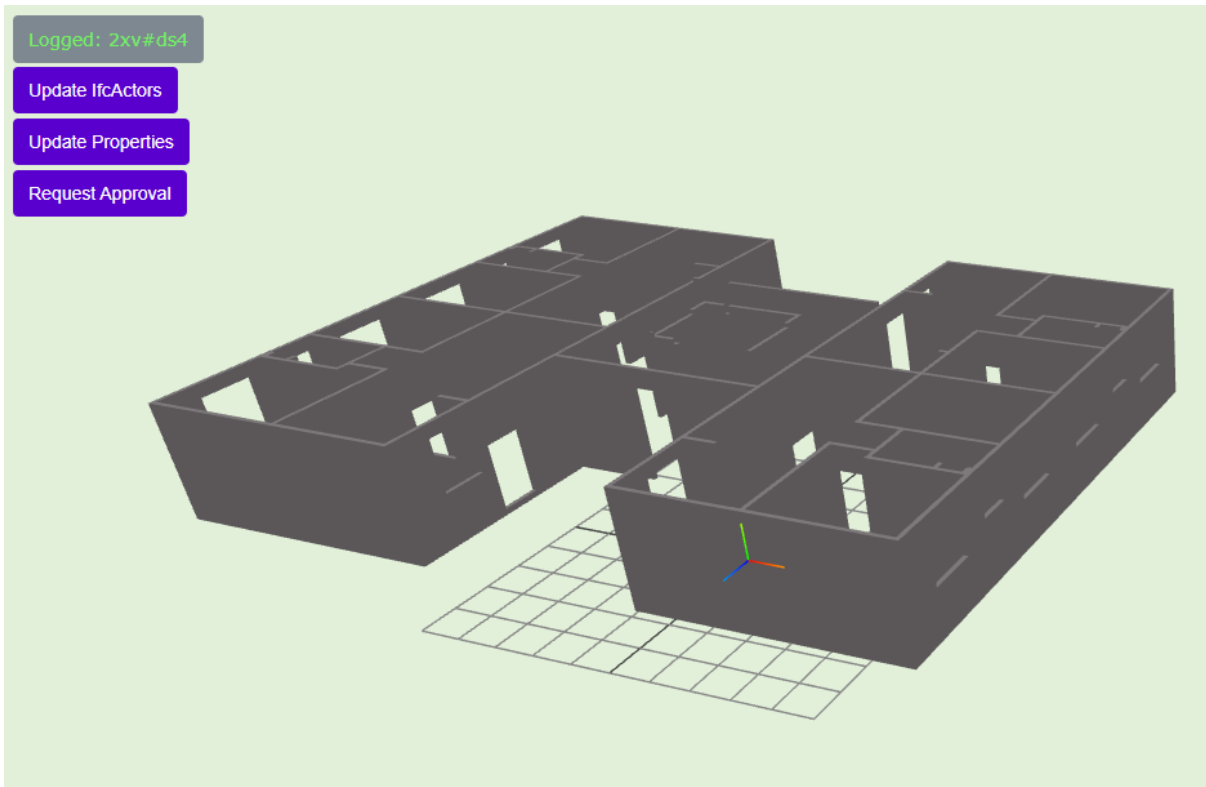


Figure 71: My Models Webpage Lead Appointed Party View.

#### 4.6.6 Collaborative Production of Information

The collaborative production of information is the stage where the appointed parties act the most. On the My Models webpage, the application provides the button Update Properties, as can be seen in Figure 72. This button will allow an update of the value of properties established on the Requirements of the IDS file. In the case of the demonstration is the *CompressiveStrenghtOnSite* property.

Therefore, the Appointed Party will select each wall, and press the Update Properties button, making available the field to enter the value. Then, it is just typing the value in the input form with what was obtained with the laboratory tests to define the concrete compressive strength used on the specific walls selected.

This process of updating the model as built could be processed in a different environment also, as an example the BEXEL Manager, even if the file is not stored in a proper common data environment. Since, the GUIDs of the elements will be maintained, and the IDS, which is responsible for defining the applicability and checking the requirements is stored on-chain, keeping track of what was defined in contract in the Tender process.

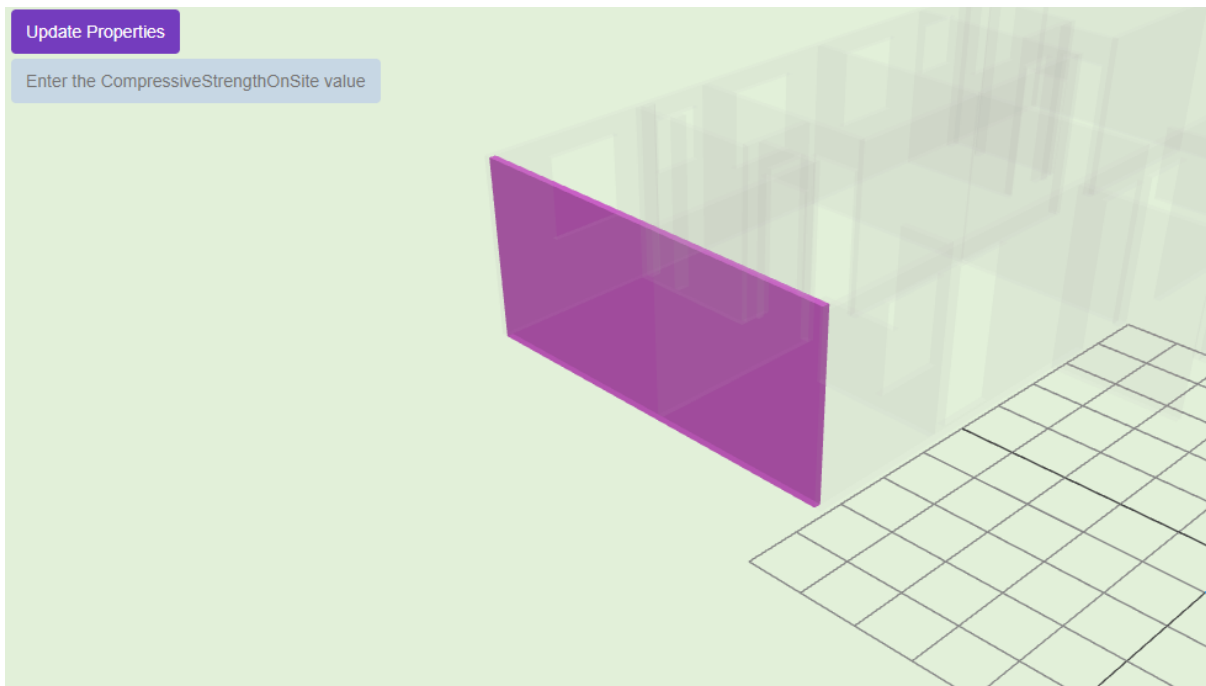


Figure 72: Updating properties.

Then, with the information produced, the last step is to request the approval of the execution. This can be performed by the Appointing Party, by clicking on the button Request Approval, which will provide the field to input a DID URL, as the Figure 73 shows.

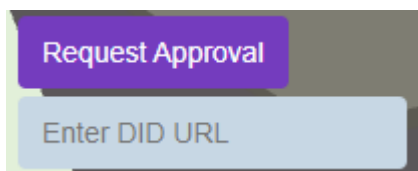


Figure 73: Request Approval Button

In the case of this demonstration, this URL can represent the DID Document with the protocols to prove the authenticity of the data imputed about the compressive strength, by checking proofs with Issuer, the laboratory the ran the analysis. The command is concluded by entering the DID URL, as it is in Figure 74.

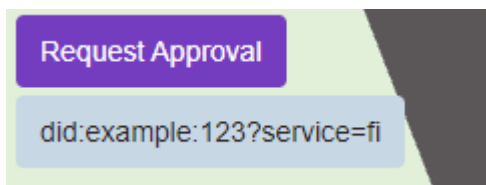


Figure 74: Connecting the DID URL



#### 4.6.7 Information Model Delivery

The process ran by the Appointed Party will call the shared function *requestApproval()*. Therefore, it will bring a notification to the Appointing Party to check the service and give the approval. This actor can also access the My Models webpage with an additional function to provide the approval, as it can be seen in Figure 75.

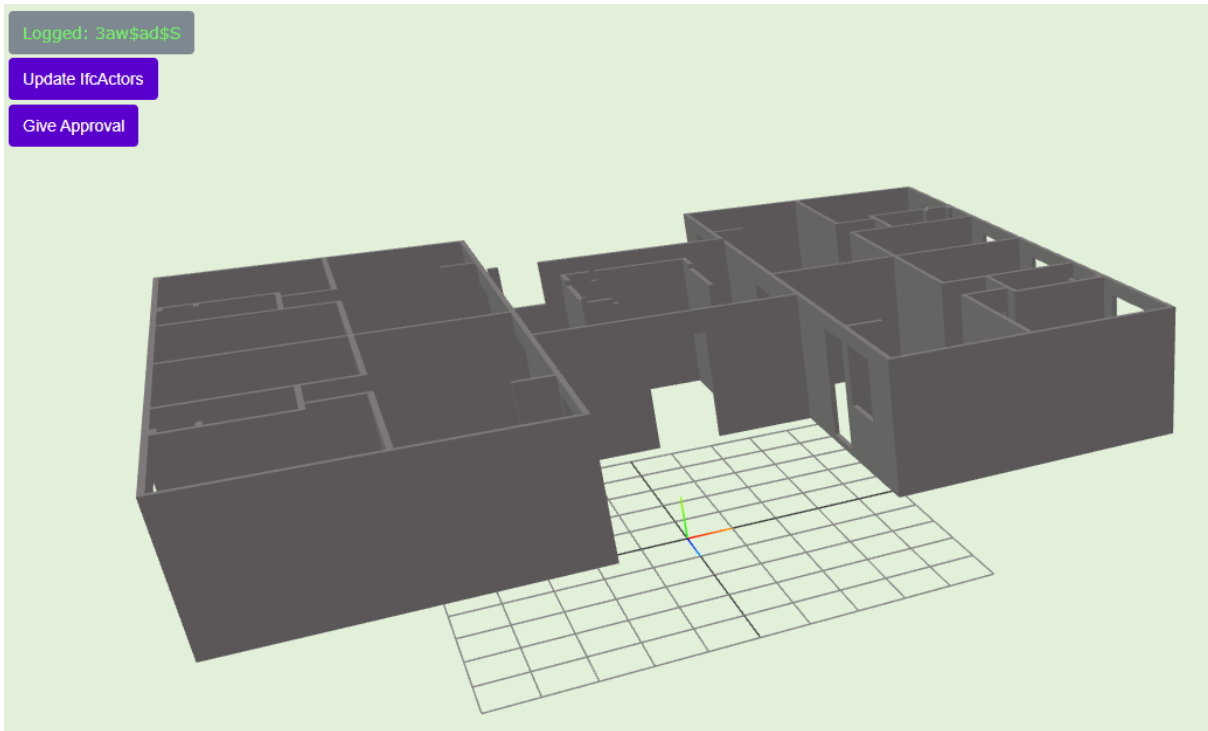


Figure 75: My Models webpage with the Appointing Party View.

#### 4.6.8 Project Close-out

Therefore, the last step, after checking the execution, is to press the button Give Approval, which will enable type a score for the execution. This score would define the quality of the work done, considering also the complexity established by the BIM Coins target. This process, associated with the connection with a Self-Sovereign Wallet, to hold this data and the NFT as badge of the service completed, would establish a reputation system, that recursively, can be used as an acceptance parameter in the Tendering.

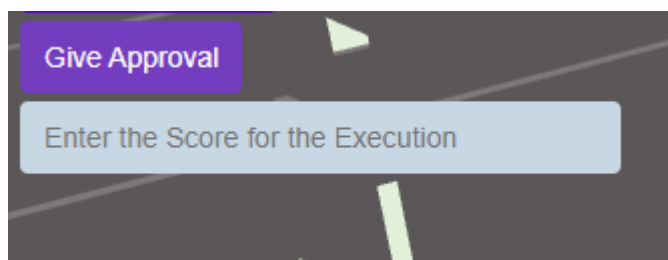


Figure 76: Giving Reputation to the Service

## 5 RESULTS AND DISCUSSION

The discussion of the results achieved will be broken down into three parts. The first part is focused on the features available for the users, highlighting its strengths and gaps, comparing to other tools. The second part relies on the effectiveness of the workflows defined and technologies used on Decentralized Marketplace. To close the discussion, an analysis is made on how some of the processes implemented could add value for value for the BUILDCHAIN Project. Therefore, with these three steps, it will be clarified the level of achievement considering the initial objectives for this work.

### 5.1 Usability Assessment

The step-by-step use of the application demonstrated in the section 4.6 highlighted the simplicity generated for the end user to perform complex operations in the background, like writing new data on top of the IFC file. Although, it also shown that there are several features that should be implemented yet, to fully cover real world tendering processes for construction services and other AEC related projects.

The major strength of the application is to give access to only the data that matters for a specific actor, and it is clearly demonstrated by the usability of the platform. When comparing the whole model in Figure 64 with the subset of data in Figure 71, the difference is pointed out. This mechanism performed through the IDS data brings transparency of the clients' needs and requirements, and assertiveness in the delivery of new information by the appointed party.

In the stage 5.1 of Assessment and Need is where the Application lacks the most in terms of usability. The user must fully build the IDS file outside the application, which can be complex since it is an XML file. Another relevant point is that should be implemented an option to create new IDS files by selecting partial information from a database of projects from the Client, as shown in Figure 77. This gap gets relevant, especially when compared to a platform available in the market called Plannerly. This platform is a very powerful tool for ensuring data quality in the projects and was the first commercial software to implement the export of IDS files (Plannerly, 2023). Which is done in a very clever way that enables the user to structure the IDS seamlessly while he is planning and defining the scope of the project. Although, a huge benefit of the Application built in this comparison is the decentralization of the Information Delivery Specification, which reflects in the usability as well by increasing trust in the definitions. Therefore, both applications can be seen as complementary in many aspects.

The last point to be discussed in the Usability Assessment is the impact of enabling IFC data update directly on the Web, without opening any proprietary software. The feature implemented enables updating properties that might be defined in the requirements of the IDS. Considering the Construction phase, that is the focus of this work, in general there will be models already built in the design phase,

therefore, the main process will be updating properties according to results on field, which this functionality can cover satisfactorily. To highlight the relevancy of this tool, a similar feature could be implemented in the Autodesk environment using the Design Automation API, which enables automated modifications in Revit files by processing the files on the Autodesk cloud, to run a process like this, it would cost an average of 0,10 dollars per minute (Nagy et al., 2020). While, in the feature implemented with the IFC.js opensource library, the modifications are performed already on IFC files, in the browser, completely free, since require no cloud power.

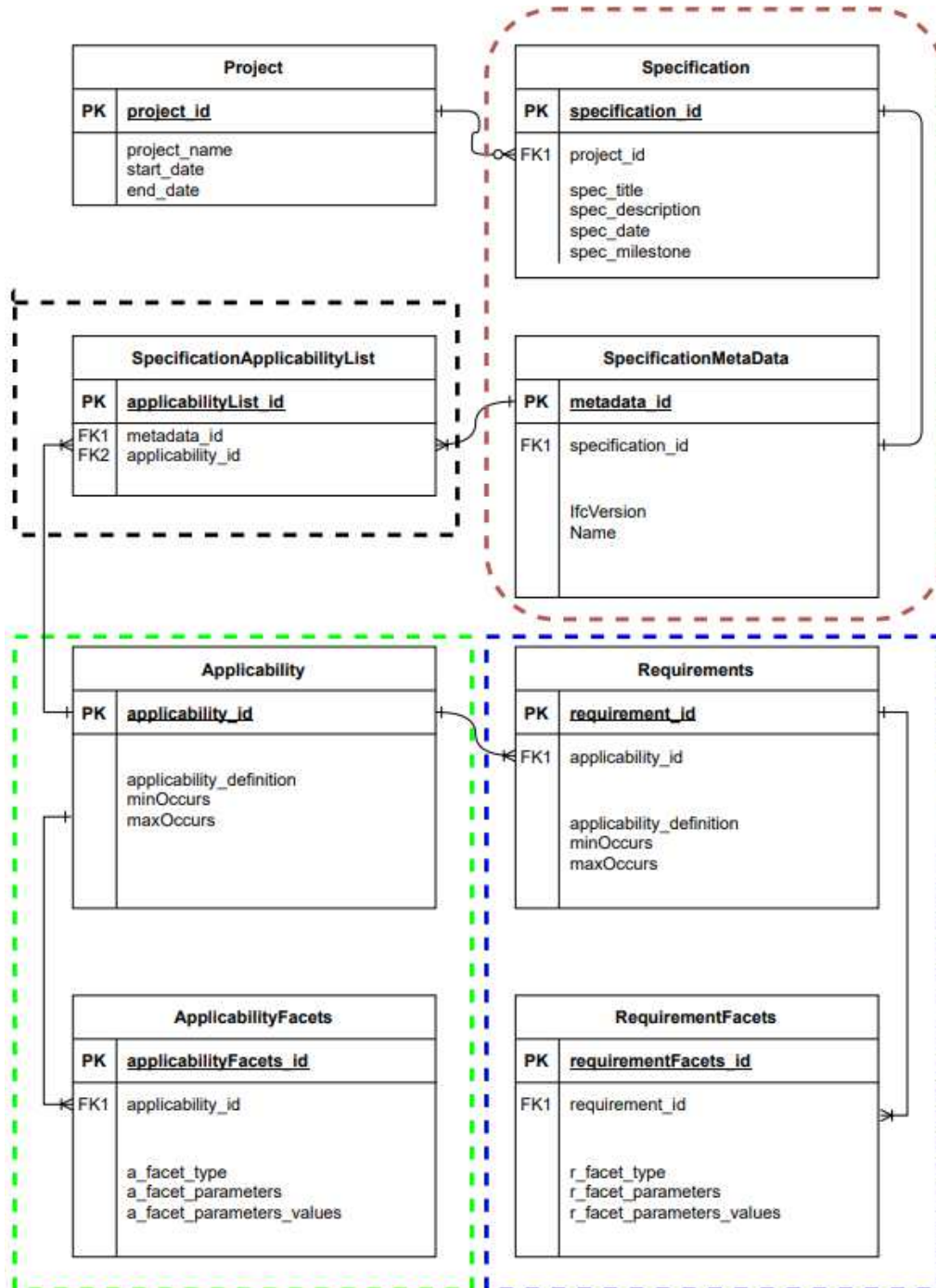


Figure 77: Design possibility for a IDS database

## 5.2 Discussion on the Effectiveness

The first point to highlight in this discussion is the data federation with consistency and cost effectiveness that the Decentralized Application brings. The ability to store the IDS data on chain enabled using the Internet Computer turns out to be a scalable process considering the costs highlighted in section 4.1. Thereby, this mechanism can be extensively used to require information and specify deliverables from sources that are stored in different databases, as it was demonstrated in section 4.6.

It is important to point out also the effectiveness of the Reputation System proposed running in the backend. It was designed a recursive process to benefit the construction service execution quality. The general concept of the system is outputted in Figure 78, in a workflow that the Client establishes the complexity of the work in the moment of making it available in the marketplace, and then, after the work is finished, it is given a score for the quality of the execution. Therefore, the Contractor can use this metadata as part of the proofs of experience in a future tender process, even considering new Clients, since, this is decentralized information, auditable on chain.

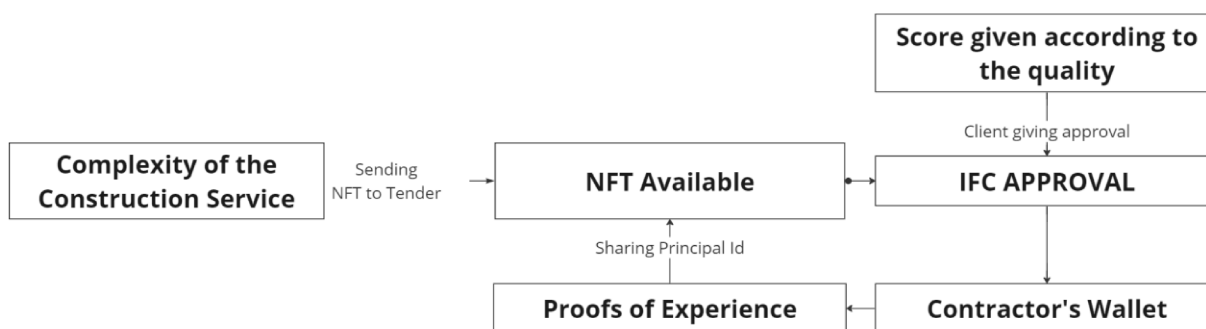


Figure 78: Reputation System Workflow

Through the use of smart contracts, decentralized identifiers, and blockchain technology, the Decentralized Marketplace produces trustworthy information. Transparent, pre-established rules governing transactions and project milestones are carried out via smart contracts. The *OpenBIM\_D* handles the transactions and interactions with users and IFC data, while the *NFTActor* is specialized in store the data of the IDS, with the communication of between both types of Smart Contract, it becomes an effective way to achieve the needs. Additionally, the unchangeable ledger of the blockchain captures all transactions and interactions, offering an auditable history that promotes confidence among all parties. In this ecosystem, trust is established by accessible automation, cryptographic security, and an unchangeable record of actions, ensuring the accuracy of the data in the marketplace.

Although, it is important to underline that some functions of the smart contract could be automated to be executed based on real-time data and verification from IoT sensors, ensuring compliance and eliminating the need for intermediaries, but the application is not effective in this point right now. As an example, the process of sharing the actual compressive strength of the concrete walls, could be done

using IoT devices, in a way that the manufactures of these devices would be the **issuers** of the credentials, the walls related to the Approval process would be the **subjects**, and the contractors the **holders** of the verifiable credentials.

### 5.3 Added Values to the BUILDCHAIN Project

The BUILDCHAIN Project, as detailed in section 2.4, aims to deal with 12 complex use cases, and each of them must handle completely different types of data, with several needs and sources, that can go even beyond the IFC Schema. Therefore, one of the main obstacles to achieve the goals is on how standardize a process to request information in order to be logged in the Digital Building Logbook.

It is added even more complexity to this process when it is considered data coming from BIM models, which follow different standards in the information delivery. Therefore, the question left is which property from the model should be gotten to be logged in according to a use case. Thereby, the biggest value that can be added to the project is the decentralization of the Information Delivery Specification. Since, as already discussed before, this process brings the ability to specify how to request specialized data from IFC, which can be customized according to use case and the structure of the BIM model. In Figure 79, it is shown an example of a simplified high-level ontology considering some the BUILDCHAIN use cases and the workflow of this research. Although, it could be also adapted, to instead of using NFTs, store the IDS in a decentralized knowledge graph, since, in many cases, there is no Tendering involved.

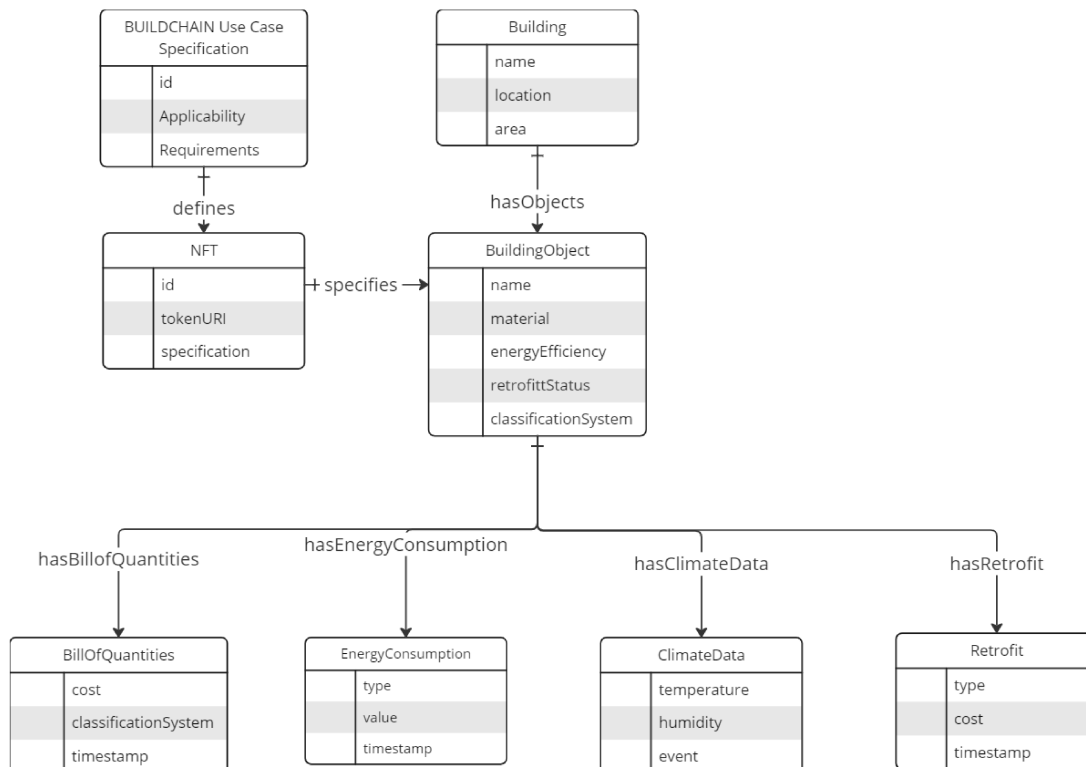


Figure 79: High Level Ontology to Specify BUILDCHAIN Use Cases Metadata

## 6 CONCLUSION AND FUTURE WORKS

The Decentralized Marketplace for Construction Services presented in this work is a complex application that relies on cutting edge technologies and advanced BIM concepts. What was presented, represents a minimum valuable product of this application can be, to infer the possible contributions by the early results achieved.

Thus, this chapter is divided into two parts, the first describing the limitations in implementation that the application faces now and the several technologies that could be used to maximize the potential, and the final part reinforces the main contributions considering the initial objectives of this work.

### 6.1 Limitations and Future Works

In terms of the limitations that the application has, it deserves to be pointed out some features that are not fully implemented yet in the IDS Parser library that was created. As it was previously detailed in section 4.2, it has not been developed yet any type of complex restrictions: Enumeration, Pattern, Bounds, Length. There are also some facets parameters to be implemented yet, presented in **Table 7**. These are elements that will be developed in future works, since there are no limitations in the IFC.js library to implement the needed functions. Thus, another relevant point that should be done, after the fully implementation, is to run the test cases that are required by buildingSMART in the developer-guide (buildingSMART International, 2022).

**Table 7:** Limitations of the IDS Parser

| <b>Facet</b>   | <b>Missing Parameter</b> |
|----------------|--------------------------|
| Entity         | Predefined Type          |
| Attribute      | -                        |
| Material       | URI                      |
| Classification | URI                      |
| Property       | Data Type, URI           |
| PartOf         | Entity without Relation  |

Another relevant limitation is regarding the BIM data transfer methods. The application uses mainly the type TM3.1, described in Figure 13 in the processes when the IFC file is loaded by IFC.js, although, the step to obtain the IFC file from the CDE or to upload it from a local repository, is a TM1.1 type since is just a function to get the array of bytes of the file, which affects the performance of the application by the automated download and upload, and also, generates more risk for outdated information.

Therefore, ideally, the solution for this problem would be migrating this file-based approach to a granular data sharing process. This could be performed, as an example, by using the Data Exchange API, from Autodesk Platform Services, to get partial data from models stored in the Autodesk Construction Cloud, but this type of process is not widely spread among the main Common Data Environments available in the market.

In the following sub-sections, it is detailed how other technologies could support and maximize the applicability and the development of new features for the Decentralized Marketplace in future works.

### **6.1.1 Integration with ONTOCHAIN framework**

The Ontologies layer of ONTOCHAIN can bring several benefits in the implementation of the decentralized marketplace. Since, it has already developed solutions like a semantically enriched distributed oracles system by the ONTOCHAIN-funded DESMO-LD project and a semantic graph to NFT data modeling by the ONTOCHAIN-funded NFTWatch project, which defines a new ontology for annotating all NFT properties (Papaioannou et al., 2022).

The distributed oracles system with semantic enrichment might facilitate seamless data interchange and communication between various IoT systems and devices utilized in construction projects. Finally, the modeling of NFT data into a semantic graph may guarantee that NFT attributes linked to trustworthiness and confidence are well-documented and understandable within the application, promoting transparency and trust among users.

Also, the SDK implemented in the Core Protocols Layer for Identity Management could enrich the processes of integrating DIDs, SSIs, and VCs with the IFC entities, therefore, the elements that represent real-world objects and processes. Thus, the use of the ONTOCHAIN framework could be highly beneficial, since one of the most important limitations at this moment in the DApp, is the deployment of a semantic graph that could better link the NFT data, *IfcApproval* and DIDs properties, especially, considering the discussions about ontology approaches for BUILDCHAIN.

### **6.1.2 IoT, Edge-computing, and Containerization**

There are many advantages to integrating IoT through edge computing into the Decentralized Marketplace for building services. IoT sensors on building sites continuously keep an eye on the machinery, safety conditions, and environmental factors while providing the market with data in real time. Stakeholders are better equipped to quickly make informed decisions because of this data transparency.

Edge computing lowers the latency of data processing, allowing for quick reactions to equipment problems and safety concerns. Predictive maintenance algorithms can deal with equipment issues before

they become serious, increasing project efficiency, and decreasing downtime. Then, the Marketplace smart contracts may be triggered by IoT data. For instance, money can be automatically transferred upon reaching predetermined project milestones, expediting transactions, and guaranteeing contract compliance.

In essence, the application is improved by real-time data, predictive capacities, and automation thanks to IoT integration through edge computing. The construction business becomes more connected and intelligent as a result, encouraging openness, efficiency, safety, and cost-effectiveness in projects. This implementation would be crucial to the DID process discussed in section 5.2.

Kubernetes and other container orchestration solutions make automated management, monitoring, and upgrades possible, which improves the application's overall efficiency, robustness, and scalability. Also, the application could become highly portable and consistent across various settings by being encapsulated in containers along with its dependencies.

### **6.1.3 Integration with buildingSMART Data Dictionary (bSDD) and BIM Collaboration Format (BCF)**

The connection between these other technologies from buildingSMART, the Data Dictionary, and BIM Collaboration Format is a must for this application since they perfectly fit some of the workflows and both have structured APIs, which would enable this integration. Both tools are already supposed to be correlated with the use of IDS, as shown in Figure 11.

In the proposed workflow by buildingSMART, the bSDD is used to provide data enrichment as it enables to find the right classifications, properties, and allowed values. Therefore, it could be implemented in this application in the Assessment and Need stage to support the Appointing Party to better define the information requirements, as well as it could be implemented in the Collaborative Production of Information to help the Appointed Party to update the properties correctly.

The BIM Collaboration Format (BCF) is defined to be used in the validation of data against the IDS, to export the issues that were found. Thus, it could be implemented in the application to be used during the Approval processes. Therefore, standardizing the collaboration approach during the use of the application.

## **6.2 Contributions and Final Thoughts**

Bringing back the initial objectives of this research which were to design processes to provide more transparency for Tendering, to establish workflows for specifying information deliverables to the BUILDCHAIN use cases, and to provide trustworthiness to data linked along the projects' lifecycle, it



is possible to output several contributions of this work based on the evidence shown throughout the sections.

The most remarkable point is the approach of storing the IDS file on chain, providing a scalable and customizable way to request and check data from IFC Models, using decentralized technologies. This process can be used like a kernel, to define which data goes to Digital Building LogBook, according to the requirements of the use case and the modeling standards. It also has a high fit to check information requirements during the design phase of the projects.

Another contribution that should be pointed out is the mechanism to write the decentralized identities directly on the IFC files, enabled by the IFC.js library. This process sustains a bigger approach to provide and assign the right data, to the right person, at the right time.

A final contribution of this research must be making open source the *IDSParser* functions developed on top of the IFC.js and IFC Schema. This should be done, as soon as all the facets are fully implemented, and the application passes in all the test cases proposed by buildingSMART. The IDS standard is becoming recognized as a powerful mechanism for AEC workflows; therefore, this contribution would amplify the use of the standard providing one tool based on JavaScript and supporting the openBIM community.

## REFERENCES

- Afsari, K., Eastman, C. M., & Castro-Lacouture, D. (2017). JavaScript Object Notation (JSON) data serialization for IFC schema in web-based BIM data exchange. *Automation in Construction*, 77, 24–51. <https://doi.org/10.1016/j.autcon.2017.01.011>
- Berlo, L. (2019). *The curious case of the MVD*. <https://www.buildingsmart.org/the-curious-case-of-the-mvd/>
- BUILDCHAIN*. (2022). <https://buildchain-project.eu/>
- buildingSmart International. (n.d.). *ifcOWL*. Retrieved July 24, 2023, from <https://technical.buildingsmart.org/standards/ifc/ifc-formats/ifcowl/>
- buildingSmart International. (2017). *IFC 4.0.2.1 Standard*. [https://standards.buildingsmart.org/IFC/RELEASE/IFC4/ADD2\\_TC1/HTML/](https://standards.buildingsmart.org/IFC/RELEASE/IFC4/ADD2_TC1/HTML/)
- buildingSMART International. (2019). *IFC4\_ADD2\_TC1 Documentation*. IFC4\_ADD2\_TC1.
- buildingSMART International. (2022). *IDS Documentation*. <https://github.com/buildingSMART/IDS/tree/master/Documentation>
- Casillo, M., Colace, F., Gupta, B. B., Lorusso, A., Marongiu, F., & Santaniello, D. (2022). Blockchain and NFT: a novel approach to support BIM and Architectural Design. *2022 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT)*, 616–620.
- Cocco, L., Tonelli, R., & Marchesi, M. (2022). A System Proposal for Information Management in Building Sector Based on BIM, SSI, IoT and Blockchain. *Future Internet*, 14(5). <https://doi.org/10.3390/fi14050140>
- Donkers Alex and Yang, D. and de V. B. and B. N. (2023). A Visual Support Tool for Decision-Making over Federated Building Information. In C. and R. A. Turrin Michela and Andriotis (Ed.), *Computer-Aided Architectural Design. INTERCONNECTIONS: Co-computing Beyond Boundaries* (pp. 485–500). Springer Nature Switzerland.
- Erri Pradeep, A. S., Yiu, T. W., & Amor, R. (2019). Leveraging blockchain technology in a bim workflow: A literature review. *International Conference on Smart Infrastructure and Construction 2019, ICSIC 2019: Driving Data-Informed Decision-Making*, 371–380. <https://doi.org/10.1680/icsic.64669.371>

- Greenwald, N., Sanford, E., & Bim, V. / . (2020). *BIM, Blockchain & Smart Contracts* (Vol. 40, Issue 4).
- IFCJS. (2022). *IFCJS Documentation*. <https://ifcjs.github.io/info/docs/introduction>
- Internet Computer. (2022). *Internet Computer, ICP, How it works*. <https://internetcomputer.org/how-it-works?source=nav>
- ISO 16739-1:2018 Industry Foundation Classes (IFC) for data sharing in the construction and facility management industries — Part 1: Data schema. (2018). In *ISO 16739-1: Vol. ISO 16739-1*. ISO.
- Jiang, S., Jiang, L., Han, Y., Wu, Z., & Wang, N. (2019). OpenBIM: An enabling solution for information interoperability. In *Applied Sciences (Switzerland)* (Vol. 9, Issue 24). MDPI AG. <https://doi.org/10.3390/app9245358>
- Kochovski, P., & Stankovski, V. (2021). Building applications for smart and safe construction with the DECENTER Fog Computing and Brokerage Platform. *Automation in Construction*, 124. <https://doi.org/10.1016/j.autcon.2021.103562>
- Maratkar, P. S., & Adkar, P. (2021). React JS-An Emerging Frontend JavaScript Library. *Iconic Research And Engineering Journals*, 4(12), 99–102.
- Nagy, A., Gonçalves, A., & Patil, V. (2020). *Estimate Design Automation costs*. <https://aps.autodesk.com/blog/estimate-design-automation-costs>
- Papaioannou, T. G., Kochovski, P., Shkempi, K., Barelle, C., Simonet-Boulogne, A., Ciaramella, M., Ciaramella, A., & Stankovski, V. (2022). *A Blockchain-based, Semantically-enriched Software Framework for Trustworthy Decentralized Applications*. <https://contractfortheweb.org/>
- Pattini, G., Locatelli, M., Tagliabue, L. C., & Di Giuda, G. M. (2022). A theoretical framework for automatic BIM validation and smart contracts in the design phase. *EC3 Conference 2022*, 3, 0.
- Pauwels, P., de Koning, R., Hendriks, B., & Torta, E. (2023). Live semantic data from building digital twins for robot navigation: Overview of data transfer methods. *Advanced Engineering Informatics*, 56. <https://doi.org/10.1016/j.aei.2023.101959>
- Pauwels, P., & Terkaj, W. (2016). EXPRESS to OWL for construction industry: Towards a recommendable and usable ifcOWL ontology. *Automation in Construction*, 63, 100–133. <https://doi.org/10.1016/j.autcon.2015.12.003>
- Plannerly. (2023). *How to Implement IDS XML in your BIM workflow*. <https://plannerly.com/ids-xml-how-to/>

- Preukschat, A., Reed, D., Allen, C., & Vogelsteller, F. (2021). *Self-Sovereign Identity Decentralized Digital Identity and Verifiable Credentials*.
- Russo, C. (2022). *Motoko Language*. <https://internetcomputer.org/docs/current/motoko/main/motoko>
- Shoup, V. (2022). *The Internet Computer for Geeks*. <https://dfinity.org/foundation/>;
- Taelman, R., Van Herwegen, J., Vander Sande, M., & Verborgh, R. (2018). Comunica: a Modular SPARQL Query Engine for the Web. *Taelman\_iswc\_resources\_comunica\_2018*. <https://comunica.github.io/Article-ISWC2018-Resource/>
- Tao, X., Wu, Z., Zheng, C., Liu, H., Das, M., & Cheng, J. C. P. (2021). *Knowledge graph-driven smart contract for metadata checking in blockchain-based BIM collaboration*.
- Teisserenc, B., & Sepasgozar, S. M. E. (2022). Software Architecture and Non-Fungible Tokens for Digital Twin Decentralized Applications in the Built Environment. *Buildings*, 12(9), 1447.
- Tomczak, A., Berlo, L. V., Krijnen, T., Borrmann, A., & Bolpagni, M. (2022). A review of methods to specify information requirements in digital construction projects. *IOP Conference Series: Earth and Environmental Science*, 1101(9). <https://doi.org/10.1088/1755-1315/1101/9/092024>
- UK BIM Framework. (2022a). *ISO 19650 Guidance 2: Delivery phase*. <https://ukbimframeworkguidance.notion.site/ISO-19650-Guidance-2-Delivery-phase-6124641a84d64bd09d30ad57a506629f>
- UK BIM Framework. (2022b). *ISO 19650 Guidance D: Developing information requirements*. <https://ukbimframeworkguidance.notion.site/ISO-19650-Guidance-D-Developing-information-requirements-4735648bd36f468d8694a25d44f94bfc>
- Yu, A. (2022). *The Complete Web Development Course*. <https://appbrewery.com/p/the-complete-web-development-course>