Multimedia agents using MPEG-21 contracts and blockchain smart contracts

A Degree Thesis
Submitted to the Faculty of the
Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona
Universitat Politècnica de Catalunya
by
Carles Bertran Bonet

In partial fulfilment
of the requirements for the degree in Telecommunication Technologies and Services ENGINEERING

Advisor: Jaime Delgado Mercé

Barcelona, May 2021
Abstract

The development of the new standard CEL MPEG-21 has been one step forward in the digitalization of traditional text contracts related to Multimedia objects. This has opened new ways to interact with them and integrate them in already useful digital environments.

The project aims to use the information in these CEL MPEG-21 contracts to determine if an action can be allowed or not and keep actions related to that contract in a Blockchain through a smart contract.

In order to achieve these goals, software has been developed and the internal architecture designed. The project includes a simulation of the software tests. After the testing it can be affirmed that the software is capable of extracting and applying all the data inside the CEL MPEG-21 contracts and then store them in a smart contract deployed in the Blockchain.

The software is the first step to automatize all the previous processes that were made using the traditional text contracts.
**Resum**

El desenvolupament del nou estàndard CEL MPEG-21 ha millorat la digitalització dels contractes de text tradicionals relacionats a continguts multimèdia. Això ha obert noves vies per interactuar amb aquests contractes i integrar-los en entorns digitals ja funcionals.

L'objectiu principal del projecte és utilitzar la informació en els contractes CEL MPEG-21 per determinar si una acció es pot realitzar o no i registrar les accions relacionades amb el contracte en una Blockchain a través de un smart contract.

Per aconseguir aquest objectiu, s'ha desenvolupat software i dissenyat la arquitectura interna. El projecte inclou la simulació de les proves de software. Després de les proves, es pot afirmar que el software és capaç de extreure i aplicar la informació compresa en els contractes CEL MPEG-21 i, llavors, emmagatzemar les accions en un smart contract desplegat en la Blockchain.

El software és el primer pas per automatitzar els processos que prèviament requerien l’ús dels contractes tradicionals de text.
Resumen

El desarrollo de un nuevo estándar CEL MPEG-21 ha mejorado la digitalización de los contratos de texto tradicionales relacionados a contenidos multimedia. Esto ha abierto nuevas posibilidades para interactuar con estos contratos e integrarlos en entornos digitales ya funcionales.

El objetivo principal del proyecto es utilizar la información en los contratos CEL MPEG-21 para determinar si una acción se puede realizar o no, y registrar las acciones relacionadas con el contrato en una Blockchain a través de un Smart contract.

Para lograr este objetivo, se ha desarrollado software y diseñado la arquitectura interna. El proyecto incluye la simulación de las pruebas realizadas con el software. Una vez hechas las pruebas, se puede afirmar que el software es capaz de extraer y aplicar la información contenida en los contratos CEL MPEG-21 y entonces almacenar las acciones en un Smart contract desplegado en la Blockchain.

El software es el primer paso para automatizar los procesos que previamente requerían el uso de los contratos tradicionales de texto.
Acknowledgements

My most sincerely thanks to my director Jaime Delgado for his support and the opportunity given to collaborate with his project. Special thanks to ETSETB for everything that lead me to this moment. Finally, thanks to my family for their support and patience.

My last mention to my future wife Sandra, she is the light that guides me when I am lost.
## Revision history and approval record

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>25/03/2021</td>
<td>Document creation</td>
</tr>
<tr>
<td>1</td>
<td>01/04/2021</td>
<td>Document revision</td>
</tr>
<tr>
<td>2</td>
<td>15/04/2021</td>
<td>Document revision</td>
</tr>
<tr>
<td>3</td>
<td>29/04/2021</td>
<td>Document revision</td>
</tr>
<tr>
<td>4</td>
<td>07/05/2021</td>
<td>Document approved</td>
</tr>
</tbody>
</table>

## DOCUMENT DISTRIBUTION LIST

<table>
<thead>
<tr>
<th>Name</th>
<th>e-mail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Carles Bertran Bonet</td>
<td><a href="mailto:cbertri@gmail.com">cbertri@gmail.com</a></td>
</tr>
<tr>
<td>Jaime Delgado Mercé</td>
<td><a href="mailto:jaime.delgado@upc.edu">jaime.delgado@upc.edu</a></td>
</tr>
</tbody>
</table>

## written by: Reviewed and approved by:

<table>
<thead>
<tr>
<th>Date</th>
<th>Name</th>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>25/03/2021</td>
<td>Carles Bertran Bonet</td>
<td>Project Author</td>
</tr>
<tr>
<td>07/05/2021</td>
<td>Jaime Delgado Mercé</td>
<td>Project Supervisor</td>
</tr>
</tbody>
</table>
## Table of contents

Abstract ........................................................................................................................................... 1  
Resum ............................................................................................................................................... 2  
Resumen .......................................................................................................................................... 3  
Acknowledgements ......................................................................................................................... 4  
Revision history and approval record .............................................................................................. 5  
Table of contents ............................................................................................................................. 6  
List of Figures ................................................................................................................................... 8  
List of Tables: .................................................................................................................................... 10  
1. Introduction .................................................................................................................................... 11  
   1.1. Objectives of the project ............................................................................................................. 11  
   1.2. Requirements and specifications ................................................................................................. 11  
   1.3. Methods and procedures ............................................................................................................ 11  
   1.4. Deviations of the initial plan ....................................................................................................... 12  
   1.5. Workplan: Gantt diagram .......................................................................................................... 13  
2. State of the art of the technology used or applied in this thesis ..................................................... 14  
   2.1. The CEL MPEG-21 standard multimedia contracts ................................................................. 14  
   2.2. Ethereum blockchain .................................................................................................................. 15  
      2.2.1. What is Blockchain? ................................................................................................................ 15  
      2.2.2. What is Ethereum? .................................................................................................................. 15  
      2.2.3. Smart contracts in Solidity ..................................................................................................... 16  
3. Project development ....................................................................................................................... 17  
   3.1. Overall overview of the project .................................................................................................. 17  
      3.1.1. High level overview ............................................................................................................... 17  
      3.1.2. Implementation overview ...................................................................................................... 18  
         3.1.2.1. Frontend of the Media Agent ............................................................................................. 18  
         3.1.2.2. Backend of the Media Agent ............................................................................................ 18  
         3.1.2.3. Blockchain/Smart contract section ................................................................................... 19  
   3.2. CEL MPEG-21 contracts .......................................................................................................... 19  
   3.3. API CEL Data Parser ............................................................................................................... 21  
   3.4. Multimedia Agent Controller Frontend .................................................................................... 22  
   3.5. Multimedia Player ..................................................................................................................... 23  
   3.6. Media Agent Controller Backend ............................................................................................ 24  
   3.7. Solidity Smart Contract ............................................................................................................. 25
List of Figures

Figure 1. Solidity smart contract example ................................................................. 16
Figure 2. Project high level flow diagram ................................................................. 17
Figure 3. Flow diagram of the full project implementation ........................................ 18
Figure 4. CEL MPEG-21 Party example code ............................................................ 19
Figure 5. CEL MPEG-21 Object example code ......................................................... 20
Figure 6. CEL MPEG-21 Constraint example code ................................................... 20
Figure 7. API Parser function example code .............................................................. 21
Figure 8. Variable lists simulating a database ............................................................ 22
Figure 9. Media Player function called in the backend .............................................. 23
Figure 10. Example code of the Object check in the backend ................................. 24
Figure 11. Code processing an allowed request ......................................................... 24
Figure 12. Solidity retrieveData code function .......................................................... 25
Figure 13. Solidity storeData code function .............................................................. 26
Figure 14. Web3-Deployed contract flow diagram .................................................... 27
Figure 15. JavaScript code to construct a contract instance ...................................... 28
Figure 16. JavaScript code to read report.txt .............................................................. 28
Figure 17. JavaScript code to store data in the Blockchain ....................................... 29
Figure 18. PowerShell looper script ........................................................................ 29
Figure 19. Flow diagram of looper.ps1 ..................................................................... 30
Figure 20. Request analysis workflow ....................................................................... 31
Figure 21. Frontend request window ......................................................................... 32
Figure 22. Example completed request window ....................................................... 32
Figure 23. Example rejection window ....................................................................... 32
Figure 24. Media Player playing sample01.mp4 ....................................................... 33
Figure 25. Programming language usage in the project ............................................ 34
Figure 26. Gas used per transaction plot ................................................................... 35
Figure 27. CEL MPEG-21 contract structure ............................................................ 43
Figure 28. CEL MPEG-21 operative part structure .................................................. 44
Figure 29. CEL MPEG-21 deontic clause structure ............................................... 45
Figure 30. Ganache configuration parameters ........................................................... 48
Figure 31. Truffle compiling smart contract ............................................................... 48
Figure 32. Truffle deploying smart contract ............................................................... 49
Figure 33. Blockchain network parameters code ..................................................... 49
Figure 34. Ganache contract deployed ................................................................. 49
Figure 35. Check resultant Media player .............................................................. 50
Figure 36. Resultant report from allowed request ............................................... 51
Figure 37. Ganache storage post allowed request check ...................................... 51
Figure 38. Results window from check request 2 ............................................... 52
Figure 39. Resultant report from wrong user request ......................................... 52
Figure 40. Ganache storage post wrong user request check ................................ 52
Figure 41. Results window from check request 3 ............................................... 53
Figure 42. Resultant report from wrong video request ....................................... 53
Figure 43. Ganache storage post wrong video request check ................................ 54
Figure 44. Results window from check request 4 ............................................... 54
Figure 45. Resultant report from wrong delivery method request ....................... 54
Figure 46. Ganache storage post wrong delivery method request check ............... 55
Figure 47. Results window from check request 5 ............................................... 55
Figure 48. Resultant report from wrong mean request ....................................... 56
Figure 49. Ganache storage post wrong mean request check ................................ 56
Figure 50. Results window from check request 6 ............................................... 56
Figure 51. Resultant report from wrong date request ......................................... 57
Figure 52. Ganache storage post wrong date request check ................................ 57
Figure 53. Results window from check request 7 ............................................... 57
Figure 54. Resultant report from wrong country request .................................... 58
Figure 55. Ganache storage post wrong country request check ........................... 58
Figure 56. Results window from check request 8 ............................................... 58
Figure 57. Resultant report from wrong language request .................................. 59
Figure 58. Ganache storage post wrong language request check ....................... 59
List of Tables

Each table in the thesis must be listed in the “List of Tables” and each must be given a page number for its easy location.

Table 1. CEL Parsing API functions ................................................................. 21
Table 2. All possible allowed request .................................................................. 33
Table 3. Request already expired ....................................................................... 33
Table 4. Blockchain transaction cost .................................................................. 35
Table 5. Main goals accomplished checklist ....................................................... 36
Table 6. Costs of equipment .............................................................................. 37
Table 7. Total costs of the projects .................................................................... 38
Table 8. Allowed request to check ..................................................................... 50
Table 9. Wrong user request to check ................................................................ 52
Table 10. Wrong video request to check ............................................................ 53
Table 11. Wrong delivery method request to check ............................................ 54
Table 12. Wrong mean request to check ............................................................. 55
Table 13. Wrong date request to check ............................................................... 56
Table 14. Wrong country request to check .......................................................... 57
Table 15. Wrong language request to check ........................................................ 58
1. **Introduction**

1.1. **Objectives of the project**

The Contract Expression Language MPEG-21[1] has improved the conversion of traditional contracts to digital ones. This new format has opened a lot of new possibilities. The aim of this project is to explore some of them.

The main goals of this project are:

- To create a software utility that allows to extract the information contained in a digital contract in a format that can interact with other software.
- To develop software capable of understanding the extracted information and apply the statements inside the contract to discern if a request can be allowed or not.
- To relate the information inside the digital contracts with another environment that has immense potential for future development: the smart contracts deployed in Blockchain.
- To implement a simulation that evaluates the statements in different digital contracts and control if a user can display a multimedia content, creating a record of the interactions with this content in the blockchain.

1.2. **Requirements and specifications**

To implement the goals some criteria is stated at the beginning of the project:

- First of all, the parsing of the information inside the CEL MPEG-21 contracts has to be available for other software so it the parsing will be done in an API format.
- As the standard contains a lot of information, the project will be focused in some of them to simplify the decision tree of the software.
- The software will need to interact with a user representing a Media company so a frontend will be developed in order to made to have a demonstration of the visibility of the project.
- The communication between the Multimedia contracts agent and the smart contract software will be asynchronous to avoid complex issues that are not in the scope of the project.
- Because the most extended and used Blockchain environment, regarding smart contracts, is Ethereum, it will be the one used in this project.
- As Ethereum[2] has created a specific language called Solidity[4] to create smart contracts, the project will develop a smart contract in this language.

1.3. **Methods and procedures**

The initial idea of this projects starts from an international call made by one WG of ISO/IEC JTC1 SC29 in July 2020[5]. The goal of this call was to study the possibilities of a translation
of the MPEG-21 CEL standard contract into a smart contract deployed in Blockchain. There is more information of the call in (ISO/IEC, 2020).

At 6th of October, a lot of entities researching the call proposed their solutions. The most developed one was the response made by the Institute Mines-Télécom – Télécom SudParis IMT-TSP (Marina Ljubojevic, 2020) in which stated that a direct translation would be difficult because a smart contract is an executable code and the MPEG-21 CEL contracts are fixed XML code, so a module to interact with both of them will be needed but it was unknown how to design that module.

From this point in time, based in the solution proposed by the IMT-TSP, the project aim changed from a direct translation between the parts to the creation of an intermediary module that creates a relation between them.

1.4. Deviations of the initial plan

In the development of the Blockchain section of the project an issue was encountered regarding the environment needed to create and deploy a smart contract. The problem was that to test the validity of the code and simulate the transactions with the Blockchain, the real Ethereum network could not be used because each transaction would cost crypto money.

At this point, the development needed a testing environment free of cost. The first one to use was the remix tool implemented by Ethereum (Ethereum, 2021). However, the fact that it was web browser based but no user session existed made impossible to save the code in the tool and consult the previous transactions made by the smart contract in the Blockchain. It was also impossible to communicate that environment with the rest of the project so it was finally discarded.

At this point, it was studied if the Truffle Suite tools could be used and adapted to the project. From all the options that this suite included the project mainly required 2 of them: to compile and deploy smart contracts use the Truffle tool and to simulate the Blockchain environment and recreate Ethereum users it was intended to use the Ganache tool.

The learning time to understand how to use this tool was not included in the first planning of the project. Also, during the installation and configuration of the tools some errors were encountered related to the deployment on the Operating System that required further time to be solved.

In December the project was still in the phase of creating the Blockchain environment so, it was decided to extend the time planned for the development of the project until May.

In the Gantt diagram of the following page, it can be seen the final time allocation in the project including the time extension.
## 1.5. Workplan: Gantt diagram

### Workplan:

<table>
<thead>
<tr>
<th>September</th>
<th>October</th>
<th>November</th>
<th>December</th>
</tr>
</thead>
<tbody>
<tr>
<td>W1</td>
<td>W2</td>
<td>W3</td>
<td>W4</td>
</tr>
<tr>
<td>W5</td>
<td>W6</td>
<td>W7</td>
<td>W8</td>
</tr>
<tr>
<td>W9</td>
<td>W10</td>
<td>W11</td>
<td>W12</td>
</tr>
<tr>
<td>W13</td>
<td>W14</td>
<td>W15</td>
<td>W16</td>
</tr>
<tr>
<td>W17</td>
<td>W18</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Investigation and research

1.1 Research online MPEG-21 CEL contracts
1.2 Research online Solidity smart contracts
1.3 Research online Tezos smart contracts

### Learning and execution of contracts

2.1 Execute Solidity contracts
2.3 Realization of SmartContract tutorials

### About the international Call

3.1 Evaluate proposed solutions
3.2 Integrations possible solutions in the project

### Development of API's code

4.1 Create an API's code to parse CEL contracts
4.2 Develop a Solidity smart contract
4.3 Create a simulated Blockchain environment
4.4 Testing and analysing the code
4.5 Debugging errors from the code
4.6 Evaluation of the obtained results
4.7 Conclusion of the obtained results

### Deliveries

5.1 Realization of the proposal plan
5.2 Realization of the critical review
5.3 Realization of the final memory
5.4 Revision of the final memory
5.5 Correction of the final memory

### Deliveries:

<table>
<thead>
<tr>
<th>January</th>
<th>February</th>
<th>March</th>
<th>April</th>
<th>May</th>
</tr>
</thead>
<tbody>
<tr>
<td>W19</td>
<td>W20</td>
<td>W21</td>
<td>W22</td>
<td>W23</td>
</tr>
<tr>
<td>W24</td>
<td>W25</td>
<td>W26</td>
<td>W27</td>
<td>W28</td>
</tr>
<tr>
<td>W29</td>
<td>W30</td>
<td>W31</td>
<td>W32</td>
<td>W33</td>
</tr>
<tr>
<td>W34</td>
<td>W35</td>
<td>W36</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Deliveries:

1.1 Research online MPEG-21 CEL contracts
1.2 Research online Solidity smart contracts
1.3 Research online Tezos smart contracts
2.1 Execute Solidity contracts
2.3 Realization of SmartContract tutorials
3.1 Evaluate proposed solutions
3.2 Integrations possible solutions in the project
4.1 Create an API's code to parse CEL contracts
4.2 Develop a Solidity smart contract
4.3 Create a simulated Blockchain environment
4.4 Testing and analysing the code
4.5 Debugging errors from the code
4.6 Evaluation of the obtained results
4.7 Conclusion of the obtained results
5.1 Realization of the proposal plan
5.2 Realization of the critical review
5.3 Realization of the final memory
5.4 Revision of the final memory
5.5 Correction of the final memory
2. **State of the art of the technology used or applied in this thesis**

The project main goal is to relate somehow the information contained inside the CEL MPEG-21 contracts to the Solidity smart contracts deployed in Ethereum Blockchain. As these technologies are really relevant in the thesis, they will be explained in some detail below.

2.1. **The CEL MPEG-21 standard multimedia contracts**

The contracts regarding multimedia contents have been implemented in some virtual standards to replace the classical analogic contract. The CEL MPEG-21 is the newest standard to be used in this regard. In this section of the thesis the standard will be described from a high-level point of view to help understand the code used later.

The Standard CEL MPEG-21 contract is defined in a hierarchic XML file. This structure is based on an extensive schema where all possible useful and required information of a Media contract can be found.

As there are a lot of possibilities that can be needed to imitate the classic contract, the internal structure of the file has a rigid rule that allow to express, in an understandable way, complex situation such as obligation, royalty payments, etc.

Following such rules all variables can be classified into 3 main groups:

- **Core**: the most useful and usual information found in the Media contracts. Almost all the core variables have to be defined inside of the XML for the contract to be acceptable.

- **Ipre**: specific variables that express the exploitation of intellectual property rights. They are usually found inside more generic core variables.

- **Pane**: all the variables related to payments and notifications.

These variables are not completely independent of each other, for example: a core-subject may be referred as the person who has to be paid when the contract gets to the payment section. Because of this factor, a lot of references can be found inside of the XML file increasing the complexity of the code inside.

For more detailed information about the standard, see Appendix 1: CEL MPEG-21 Standard detail in appendices.
2.2. **Ethereum blockchain**

2.2.1. **What is Blockchain?**

In the current times, by the way the world works it is needed to produce, manage and save a lot of certified data all the time. Blockchain tries to transfer the workload generated by that data from humans, who did all the work until now, to computers.

The main problem that the computers have when dealing with certified information is that they can be hacked. The solution that Blockchain introduces to avoid that risk is through its own architecture.

Blockchain as its name indicates consists mainly in a chain composed from blocks. Inside of each block 3 parts can be found:

- **Information**: all the relevant data saved in the block also called transactions.
- **Hash**: the identification number of the block. It has to be a unique and unrepeatable number. This number is generated using the block information so if the information is changed the hash changes too.
- **Previous hash**: the hash from the last block so each block is connected to the predecessor and successor blocks.

Also, there is no concept of a unique data base because each user has a copy of the database, so if a user changes his database each other user knows about it and can invalidate those changes. This feature summed to the process of hash creation which follows an additional rule that states that all hashes have to start with a random number of 0 that changes every 2159 blocks, made blockchain impossible to hack because the certification of the information is not provided by a superior entity but from the users.

The users are also the ones that creates new blocks for the chain. To create a new block a complex mathematic problem has to be resolved. The people that use processors to resolve these problems are called “miners” and when they provide the solution the rest of the community validates it. If the solution is validated a new block is added to the chain and the miner receive a reward for his efforts.

At the end, the main goal of Blockchain is to save information with so many eyes looking at it that it becomes impossible to falsify.

2.2.2. **What is Ethereum?**

What made Blockchain use to extend massively was the invention of Bitcoin[10]. This new virtual money tried to disassociate from the conventional economic system (governs and banks) and used Blockchain architecture to achieve it.

Ethereum is the next step of Bitcoin, instead of using the transactions and blocks only as a currency it also uses them to create certified contracts. These contracts are called Smart Contracts and they use a new currency called Ether that can be only used to pay for validate smart contracts.

The smart contracts consist mainly in an executable program saved in blocks in the Ethereum Blockchain. As the contract is in the Blockchain it is immutable and decentralized and the code controls the rules accorded in the contract between the parts. This characteristic makes Smart Contracts not need a third party to secure the fulfilment of the agreements.
For example, the code can be put like if the software receives input A, then creates output A, if not, then creates output B. This cold logic working of the software makes almost impossible to scam someone by the unfulfillment by one of the parts.

Ethereum was created with the implementation of Smart contracts in mind and to achieve that a programming language for writing smart contracts was needed. Currently, Solidity has erected as the primary language on Ethereum.

### 2.2.3. Smart contracts in Solidity

A contract in the sense of Solidity is a collection of code (its functions) and data (its state) that resides at a specific address on the Ethereum blockchain. It is similar to a slot in a database that you can query an alter its value through functions defined in the code.

The following code is a simple example of a smart contract in Solidity that defines a variable, sets its value and exposes it publicly:

```solidity
pragma solidity >=0.4.16 <0.8.0;
contract SimpleStorage {
    uint storedData;
    function set(uint x) public {
        storedData = x;
    }
    function get() public view returns (uint) {
        return storedData;
    }
}
```

*Figure 1. Solidity smart contract example*

In the example contract, it can be appreciated two typical parts that a lot of languages use: the first part is the declaration of variables with a type associated because Solidity is a static language so every variable has to be clearly defined. Then, follows all the functions that interacts with the previous vars to do everything that you consider.

Solidity has only 7 basic types of variables:

- **hash**: 32 byte data chunk
- **uint**: 256 bit unsigned integer
- **int**: 256 bit signed integer
- **string32**: ascii string 32 bytes
- **address**: account identifier
- **bool**: two state value

It can also work with structs created from the basic variables above mentioned.
3. Project development

In this section of the thesis, it will be explained in detail all the parts included in the software and how they have been implemented. First of all, a more general overview of the inner workings of the projects can be found, followed by a more detailed explanation of how the workflow has been implemented.

Later, each one of the modules implemented will be thoughtfully described and some examples of the code used will be included. The modules correspond to each one of the parts of the diagram included in the point 3.2.

3.1. Overall overview of the project

3.1.1. High level overview

The inner workings of the project can be described with the figure above.

Previous to the use of the software the project requires that there are 2 parts already developed and running: The first part is the existence of CEL MPEG-21 contracts that contain the statements regulating the videos. In the case of the project, the contracts have been developed to adapt to the software simulation.

Secondly, the software supposes that a Solidity smart contract has been developed and that is deployed already in the Ethereum Blockchain. In this project the Solidity smart contract has been conceived to adapt to the software simulation and it will be deployed previously in a sandbox Blockchain to simulate the real Ethereum.

Once that these 2 parts are already done the software can start to work. First of all, a user that represents a Media Company will interact with a Media Agent to make a request. A request will define the intention of a Media company to display a video that has a CEL MPEG-21 contract regulating the permissions required to display the video and all the specifications that are needed to comply. A request has to contain all the specific information that could be required later to determine if the Media company can be allowed to display the video.

Once the request information is processed, the Agent will use an API Parser to extract the data inside the MPEG-21 CEL contracts and compare it to the request specifications. Then, it will determine, using the results of this comparison, if the request can be allowed or not.

In addition, when it arrives to a conclusion, then it will communicate and pass the request data, using an asynchronous way, with another module that manages a smart contract already deployed in the Blockchain. This second controller will finally store the information received into the Blockchain.
3.1.2. Implementation overview

The main goal of the project is to connect different technologies such as CEL MPEG-21 standard contracts and Solidity smart contracts deployed in a Blockchain, going through a code that can

In order to achieve that goal, the project consists into 3 main differentiated parts: the frontend that the user can perceive (section 3.1.2.1), the backend that interacts with the CEL MPEG-21 contracts (section 3.1.2.2) and the smart contract and Blockchain (section 3.1.2.3).

3.1.2.1. Frontend of the Media Agent

When executing the frontend, the user will open a User Interface developed in Python[11] that will ask for some information related to the request that is wanted to execute. The number of variables and data has been decided trying to implement a simple simulation but it can escalate and increase in complexity as it is needed.

Once all the request information is obtained and processed it can be the case that the request fulfills the statements in the contract, hence, it is approved. When such cases happen another frontend part will be used: The Multimedia Player that will play the video used in the simulation simulating the act approved.

3.1.2.2. Backend of the Media Agent

To process the request and determine if it is allowed or not the Backend of the Media Agent will be used. Using the functions in the API Parser made in python it will obtain the information inside the contracts and compare it to the one that the user has provided, using a hierarchically tree of checks to delve in the contract as deep as it is needed.

In all cases when processing a request when the Backend determines if it is allowed or not it will write a report in a txt file that contains all relevant data of the request to store it later in the smart contract deployed in the blockchain.
3.1.2.3. Blockchain/Smart contract section

The simulation starts from the moment that the smart contract that has been developed in Solidity is deployed in the Ethereum Blockchain. Then it will start to check every 20 seconds if the report file has any new information indicating that a request has been processed.

When that happens, a contract instance will be created and then with the JavaScript[12] functions and the Web3[13] library the smart contract deployed can be accessed to the store in the blockchain the information contained in the report. Once it is done, the report will be cleared.

3.2. CEL MPEG-21 contracts

As has been explained in more detail in the 2.1 section of the thesis, there are mainly 3 types of variables used in the CEL MPEG-21 Standard: core, ipre and pane. The contracts created as samples for the simulation in the thesis will exclude the pane variables because the goal is to create a module that can authorize the use of a multimedia content accordingly to a CEL MPEG-21 contract.

To check if allows a request from an organization that wants to use the multimedia content the simulation will only focus in the concessions, permissions, etc, but not in the economic remuneration of the actions. As the pane variables are only used to payments and notifications, they will be not needed in the CEL contracts.

Stating the conditions to allow an action involving multimedia content compels that a contract must have 3 main points correctly specified:

- The parties involved in the transaction.
- The Multimedia Object that is requested.
- The permissions and constraints accorded in the contract regarding the use of the multimedia object.

The MPEG-21 CEL contracts have been designed with these 3 main points in mind. As such to fulfil the first one all these digital contracts contain two main parties involved in the contract a licensor and a requester. The parties may be persons or organizations, but the most usual case is that a requester is a multimedia enterprise or organization. The following code is an example that can be found in the contracts:

```
<cel-core:Party id="TV3">
  <cel-core:Organization>
    <cel-core:Name>TV3 Televisio de Catalunya</cel-core:Name>
    <dc:identifier>urn:VATIN: ES11111111111</dc:identifier>
    <dc:description>The catalonian public broadcasting company</dc:description>
    <cel-core:Signatory>
      <cel-core:Name>Joan Marti</cel-core:Name>
    </cel-core:Signatory>
  </cel-core:Organization>
  <cel-core:Address>carrer de la TV3, s/n 08970 Sant Joan Despí, Barcelona, SPA IN</cel-core:Address>
```

*Figure 4. CEL MPEG-21 Party example code*
The most important variable that appears in the party section is \texttt{dc:identifier}. This variable is an identification number unique to each organization or person so it can be used in the validation to identify if the user of the software is related to a contract.

The second main point is the multimedia object requested by the user. In the MPEG-21 CEL contracts there will be always a section defining that object:

\begin{verbatim}
<cel-core:Object>
  <cel-core:Item name= "Video sample 01">
    <dii:Identifier>isan:ab123yz</dii:Identifier>
  </cel-core:Item>
</cel-core:Object>
\end{verbatim}

\textit{Figure 5. CEL MPEG-21 Object example code}

Similar to a party an object can be identified by a unique variable, in this case \texttt{dii:Identifier}, so the software can verify if the object requested by the user is related to a contract.

Finally, the permissions and constrains related to the object can be found in the corresponding section inside the MPEG-21 CEL contracts. Each permission will always consist of the Object requested, the act permitted and the constrains applied to that act. The constrains section is the one that will indicate if the user is authorized to realize the act with the multimedia content even if he is involved as a party in the contract:

\begin{verbatim}
<cel-core:Constraint>
  <cel-ipre:DeliveryModality href= "urn:mpeg:mpeg21:cel:ipre:2015 DeliveryModality#Linear" />
  <cel-ipre:Runs numberOfRuns= "10" />
  <cel-core:FactIntersection>
    <cel-ipre:Language languages= "cat" />
    <cel-ipre:SpatialContext>
      <cel-ipre:Country>ES</cel-ipre:Country>
    </cel-ipre:SpatialContext>
    <cel-ipre:TemporalContext beforeDate= "2019-04-15T00:00:00" afterDate= "2022-04-15T23:59:59" />
  </cel-core:FactIntersection>
</cel-core:Constraint>
\end{verbatim}

\textit{Figure 6. CEL MPEG-21 Constraint example code}

These constrains are important and will be needed later in the software because they will dictate if an action is allowed, in this example the act can only be done free of charge, in a linear mode, in Catalan, inside Spain and between 2019 April 15th and 2022 April 15th.
3.3. **API CEL Data Parser**

As has been detailed in point 3.2 there is a lot of critical information inside the CEL MPEG-21 contracts that will be needed to determine later if a contract is inside the allowed range or not. To obtain all this data a method to parse the contract and transform it to dynamic variables that can be used in programming is needed for the project. To resolve this issue, an API Parser has been designed in python code.

This API main goals are to obtain lists of the identifiers needed to determine if a user or a multimedia object are related to the contract and import all the permissions important variables and return them as a dict with unique keys.

An XML file is a hierarchical data format represented as a tree. Python have a really useful module that allows to navigate inside this tree and the different branches that contains. This module is `xml.etree.ElementTree`[14] and in the project the version 19.7.1 is used as it is the most recent one.

All the parser code is based on this library and the two classes used to operate inside an XML: ElementTree and Element. The first one will be used to obtain the root of XML using `tree.getroot()`. Then, the elements will be navigated using the second class inherent functions: `element.tag(“tag”), element.get(“id”) and element.find(“String”).`

The following code is an example as how the mentioned elements are used:

```python
def getContractRelation (contractFile):
    tree = ET.parse(contractFile)
    root = tree.getroot()
    relations = {}
    for child in root:
        if child.tag == "{urn:mpeg:mpeg21:cel:core:2015}ContractsRelated":
                type = contractRelation.get('type')
                idRef = contractRelation.get('contractIdref')
                relations['type'] = type
                relations['contractIdref'] = idRef
    return relations
```

*Figure 7. API Parser function example code*

In the API the functions that can be called are obtain and returning the following:

<table>
<thead>
<tr>
<th>Function</th>
<th>Behaviour</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getIdParties</code></td>
<td>Obtain and return a list of all the unique identifier keys of all the parties involved in a specified contract.</td>
</tr>
<tr>
<td><code>getParty</code></td>
<td>Obtain and return a list of all the inside variables from a specified party.</td>
</tr>
<tr>
<td><code>getContractRelation</code></td>
<td>Obtain and return all other contracts related to a specified contract.</td>
</tr>
<tr>
<td><code>getContractID</code></td>
<td>Obtain and return the contract identifier of a specified contract.</td>
</tr>
<tr>
<td><code>getPermissions</code></td>
<td>Obtain and return a list of the permissions identifiers inside a specified contract.</td>
</tr>
<tr>
<td><code>getAct</code></td>
<td>Obtain and return a list of all the acts included in a specified contract.</td>
</tr>
<tr>
<td><code>getObject</code></td>
<td>Obtain and return a list of all the variables related to a multimedia object inside a particular permission inside a specified contract.</td>
</tr>
<tr>
<td><code>getConstraint</code></td>
<td>Obtain and return a list of all the constraints applied to a particular permission inside a specified contract.</td>
</tr>
<tr>
<td><code>getMeans</code></td>
<td>Obtain and return the means to use when a constrain is of the Broadcasting delivery mode.</td>
</tr>
</tbody>
</table>

*Table 1. CEL Parsing API functions*
3.4. **Multimedia Agent Controller Frontend**

The action that initiates all the process is a request made by an unknown user. This request has to contain all the information relevant to determine if a request is allowed. To avoid having to insert all this information through a command line interface, because it is not practical, a frontend using Python has been developed.

The frontend is based in the Python library **Tkinter**\(^\text{[15]}\) which allows to create a frame and place inside different graphical options such as labels, buttons or entries to read as variables. For more information about this library, see (Python Software, 2021).

There are 2 main benefits aside from more clarity in the user interactions:

The use of dropdown menus: when the user wants to introduce some variables of the request, he has to do it through dropdown menus. This allows the developer to control the possibilities of what is defined as a variable and avoids errors like using UK instead of EN when defining a country, etc.

The buttons: a request is made and starts to be processed when a confirmation button in the interface is clicked. This provides a controllable action and can be used, if wanted, to limit the flow user request.

To define the options in the dropdown menus with all the possible characteristics at the start of the code some List are defined hardcode. Supposedly all this information would be inside a database and the frontend will make some requests to it in order to obtain the data. The List are defined simulating the responses to the queries.

The information defined this way is as follows:

```
#generate a dict that relates the video data with the isan identifier imitating a Database with this information
videosDict = {}
videosDict['sample01.mp4'] = 'isan:aa111aa'
videosDict['sample02.mov'] = 'isan:bb222bb'
videosDict['sample03.mp4'] = 'isan:cc333cc'

#create a list of the video options available imitating a Database
DeliveryModes = ['Linear', 'NonLinear', 'Broadcasting']
Means = ['None', 'Satellite', 'Internet']
Countries = ['ES', 'EN', 'IT']
Languages = ['es', 'cat', 'eng']
```

*Figure 8. Variable lists simulating a database*

Finally, a function is defined to be triggered when the confirmation button is clicked. That function obtains the request data and launches the backend to determine if a request is allowed or not. If it is the latter case, then opens a new window indicating the reason the request was rejected.
3.5. **Multimedia Player**

In the cases that the Media Agent Backend has determined that a request is allowed, the next step will be to realize the Act contained in the contract. In the simulation of the project, it is impossible to contemplate all the possibilities and the means they will require.

The simulation will state that all the request will be realized when playing a sample video. In order to simulate that a simple audio and video media player implemented in Python has been developed.

As the main goals of the project do not contemplate recreating the broadcasting systems mostly used, a free open code has been used. For the complete detailed code, see (Fitzpatrick, 2019). The code provided has been modified in certain points to adapt it to the project.

The library used in the code to construct and implement the media player is **PyQt**[16][14]. This module is based on the Qt5 framework and it is used mainly to develop complex widgets, multimedia support, 2D vector graphics.

The code is divided in a flexible model-view architecture:

- The view part (**MainWindow.py**) is used to set all the visual specifications of the frame such as the position of buttons, bars images that are wanted to appear when executing the software, etc.
- The model part (**mediaplayer.py**) is used to implement all the inner functions that will be triggered when using a button or a bar.

At the moment that the Backend process an allowed request and then launches the Multimedia Player it only has to had access to the approved multimedia object specified. To simulate that behaviour, a path to a public video folder that only has the video requested will be automatically opened. The function that is used is:

```python
def open_file_argument(self, videoPath):
    path, _ = QFileDialog.getOpenFileName(self, "Open file", videoPath,
                                           "mp3 Audio (*.mp3);mp4 Video (*.mp4);Movie files (*.mov);All files (*.*)")
    if path:
        self.playlist.addMedia(QMediaContent(QUrl.fromLocalFile(path)))
        self.model.layoutChanged.emit()
```

*Figure 9. Media Player function called in the backend*
3.6. **Media Agent Controller Backend**

The backend of the Media Agent is the core of the project to interact with the CEL MPEG-21 standard and determine if a request is allowed or not. The script was developed in Python.

The behaviour of the scripts when launched from the frontend is as follows:

First checks if the user is in any of the contracts that has stored in the contracts folder. Once it has located a contract related to the user, the script looks if that contract Media Object is the one requested. If not, returns an error.

After the user and the Media Object are checked then proceeds to check the rest of the request details such as the date, the delivery method, the means in case is a broadcasting request, the country to deliver or the video language.

All the checks with the contracts are done using the API Parser functions previously explained in point 3.3. For example, the Media Object check is done calling the function `getObject()` like this:

```python
checkObject = False
for permissionID in permissionsInContract:
    objectInContract = getObject(contract, permissionID)
    if objectInContract.get(permissionID+"idObject") == videosDict.get(videoToPlay):
        checkObject = True
```

*Figure 10. Example code of the Object check in the backend*

Once all the checks are passed the request is allowed and then the script proceeds to clear the public video folder, copy the requested video inside, write the request information in the report.txt and finally launch the Media Player to display the Video.

```python
#Clean public videos folder
files = glob.glob('./PublicVideos/*')
for f in files:
    os.remove(f)

#Store video to play in public videos folder
shutil.copy('./Videos/' + videoToPlay, './PublicVideos/')

#Update the report.txt
writeReport(userID, videoToPlay, DeliveryModality, CounUser, LangUser, 'APPROVED')

#Launch the Video Player
executePlayer("./PublicVideos")
```

*Figure 11. Code processing an allowed request*

In any other case, the scripts determine the request as not allowed and proceeds to write inside the report.txt the details of the operation and returns an error indicating the reason of rejection.
3.7. **Solidity Smart Contract**

One of the main goals of the thesis is to relate the MPEG-21 standard contract with the smart contracts deployed in the blockchain. As it has been in point 1.4 a direct translation is not possible but it is possible to store important data contained initially in the CEL contract after it has been parsed and used by the controller to determine if a request from a user is allowed.

To reach that goal it has been analysed which information can be critical and may be worth to store in the Blockchain. The conclusion is that there are 3 parts important enough to store them. They respond to the following:

- Who and what: Identifications of the user and the video request.
- When, where, how, etc: all the variables used to determine if a request is allowed.
- Result and why: if a request was approved or rejected and, in the latter case, why it was rejected.

In order to store all this information, 6 Strings will be used in the contract. Blockchain and Solidity are limited in the number of variables that can be used and an error will be generated when the code needs to access a slot in the stack that is deeper than its 16th element. Because of this limitation is recommended to use the least number of variables so the contract will stop at 6.

In order to interact with a smart contract once deployed it has to have some public functions included. As the thesis goal with the Blockchain is merely store data 2 public functions are included in the smart contract:

- **retreiveData**: a getter function that it has the view characteristic making it free of the GAS consume needed in a transaction. The code of the function is as follows:

```solidity
function retrieveData() public view returns (string memory, string memory, string memory, string memory, string memory, string memory) {
    return (userID, videoID, modality, dateRequest, country, language);
}
```

*Figure 12. Solidity retrieveData code function*
An important feature of the Solidity code is that when a variable is saved or will be saved in the memory on the Blockchain a tag memory has to be added to that variable as it can be seen in the code.

- **storeData**: a setter function that obtain through arguments the new variables to store in the Blockchain. This will be the most used function when running the software of the thesis as it will be called as a cycle each loop that the report contains new information. The code of the function is the following:

```solidity
function storeData(string memory user, string memory video, string memory mode, string memory date, string memory coun, string memory res) public {
    userID = user;
    videoID = video;
    modality = mode;
    dateRequest = date;
    countryLang = coun;
    result = res;
}
```

*Figure 13. Solidity storeData code function*

### 3.8. Smart Contract JavaScript controller scripts

Once the smart contract explained in point 3.6 is deployed, the goal of the thesis is to store new information in it each time a user makes a request. The problem is that it is not possible to interact directly with a smart contract once deployed and there are only 2 ways to do it: the Ethereum API and the Web3. As the first option have a lot of complications because it is based in the JSON-RPC protocol [17] and has a complex endpoint configuration, in the thesis the second option (Web3) is used.

#### 3.8.1. Web3

Web3.js is a JavaScript library to interact with a smart contract on the Ethereum Blockchain once it has been deployed. When you use a user interface outside of the Blockchain that wants to interact with a Smart contract inside of the Blockchain somehow you have to integrate both parts. The way to achieve that integration is through Web3. A schema of the way web3 works is the following image (see ref (Shah, 2019)):
Web3 is an isomorphic library which can be used in a frontend or a backend. To use it needs access to an Ethereum Node which is a computer that runs the Ethereum Blockchain. In the thesis Ganache will be used as a Sandbox of an Ethereum Blockchain and also as an Ethereum node.

After importing and creating a connection to the Ethereum node, it is needed to create an instance of web3 that allows to configure a smart contract, passing an interface that describe all the functions that can be called from outside the blockchain. This interface is called the Application Binary Interface, from now on referred as ABI.

For a correct contract instance that the Web3 instance will create 2 things are needed to provide:

- The ABI which includes the description of all the public functions.
- The address of the smart contract.

This contract instance is the final object and the one that will be used to interact with the deployed Smart contract in the blockchain.
The following code is the one used in the JavaScript:

```javascript
const Web3 = require('web3');
const Storage = require('./build/contracts/Storage.json');
const web3 = new Web3('http://localhost:7545');
const id = await web3.eth.net.getId();
deployedNetwork = Storage.networks[id];
const contract = new web3.eth.Contract(
    Storage.abi,
    deployedNetwork.address
);
```

**Figure 15. JavaScript code to construct a contract instance**

In this code, it can be seen that first a Web3 instance (`web3`) is created connecting it to the Ethereum node (in this case to the port of Ganache running locally) and then the contract instance is created using the ABI and the Smart Contract address (`contract`).

### 3.8.2. JavaScript function (web3_setter.js)

After creating a contract instance, the script has access to all public functions described in the smart contract. At this point, there are 2 main goals remaining for the script:

- Read and obtain the request data stored in the report.txt file and then clear the file.
- Call the `storeData` function of the contract and save the information in the Blockchain.

To solve the first point, JavaScript has an API called File System (fs) that allows read and write on an existing file. The code used is:

```javascript
const readInterface = readline.createInterface({
    input: fs.createReadStream('./test/report.txt'),
    output: process.stdout,
    console: false
});
function deleteFile (){
    fs.writeFile('./test/report.txt','',(err) => {
        console.log("file cleared");
    });
}
var arguments = []
readInterface.on('line', (line) => {
    arguments.push(line);
}).on('close', () => {
    deleteFile();
    return arguments;
});
```

**Figure 16. JavaScript code to read report.txt**
The issue when reading the file is that the final goal requires to save each line as an independent variable. There are a lot of solutions to solve that, in this case it has been chosen to create a read stream, then read each line and append it as a String in an array and, finally, when detecting the End of File overwrite the file and return the array with all the information.

Once the script has obtained all the relevant data in the array arguments it is the moment to call the `storeData` function from the deployed contract. Because this call will send data and change the Blockchain is a transaction that will consume Gas:

```javascript
const init = async () => {
    const result = await contract.methods.storeData(encodeURIComponent(arguments[0]),encodeURIComponent(arguments[1]),encodeURIComponent(arguments[2]),encodeURIComponent(arguments[3]),encodeURIComponent(arguments[4]),encodeURIComponent(arguments[5])).send({from: '0xB825B32b3aA8F1aA0AAeC1B8692172EC51a17fb6', gas:3000000});
}
```

*Figure 17. JavaScript code to store data in the Blockchain*

The send method of the contract instance is always used in asynchrony, and it have to be specified which Ethereum user will pay for the transaction. In the case of the thesis is just a user generated by the Ganache. Also, as it is a more costly transaction than the more usual ones (transfers of crypto money) it can surpass the gas limit stated by the Ganache so an upper threshold is specified in the command.

### 3.8.3. Loop script (looper.ps1)

In order to not need a human resource that executes the script to update the blockchain each time a user request is processed a short script is developed in PowerShell\[18\] to control the timing of the execution of the script.

This script consists in checking every 20 seconds if report.txt contains anything. If that is the case, then it proceeds to call the JavaScript function `web3_setter.js`. The code of the script is:

```powershell
while (1){
    $var1 = type "./reports/report.txt"
    if ($var1 -ne "") {
        node .\web3_setter.js
        Start-Sleep -s 5
    } else {
        Start-Sleep -s 20
    }
}
```

*Figure 18. PowerShell looper script*
4. **Results**

In this section the thesis results will be shown starting with the general software workflow, followed by the usage and application of it and finishing with some data plots showing the main data obtained. And last but not least, it will be evaluated how many of the initial goals of the project have been achieved.

4.1. **Software workflow**

After the completion of the software the resulting workflow in each execution can be analysed. As the software is stable the routine that follows is always ruled by this workflow.

The software workflow is separated in two parts:

- The storage in the Ethereum Blockchain.
- The process to determine if the request is allowed.

4.1.1. **Storage workflow**

Once the contract is deployed in the Blockchain, it is needed to launch the looper that will observe the file `report.txt` and, if it contains data, it will trigger the JavaScript `web3_setter.js` to store it in the Blockchain. As the `web3_setter.js` deletes the data of `report.txt` once it is stored a pause of 5 seconds is added to avoid checking the report file before it is cleared.

The flow diagram of this script is as follows:

![Flow diagram of looper.ps1](image)

The process will remain in the background while the software is running. This behaviour imitates a server that is checking permanently the new reports.
4.1.2. Request analysis workflow

Once a request is made by a user the software will try to determine if the request is allowed. The decision tree that always follows to achieve it is described in the following flow diagram:

This behaviour has been analysed through different user requests to test it. For more information, see Appendix 3 Detailed process of the software checking Appendix 3.
4.2. **Usage of the software**

4.2.1. **How the software works**

At this point, you can imitate the user and made a request to the software to be judged. Is in this section that the results of the software can be checked and analysed. To start launch the script Media_Agent_Frontend.py which will display this window:

![Figure 21. Frontend request window](image)

To make a request a User ID shall be typed and then select one option for each one of the fields, then press the “OK” button. For example:

![Figure 22. Example completed request window](image)

If the request is not allowed a new window popup with the reason the request have been rejected:

![Figure 23. Example rejection window](image)
In the other case, when the request is allowed the backend will launch the Media Player and open a window to the user to select the Multimedia Object from the Public Video folder. Once selected it will be displayed in the playlist of the Media Player:

![Figure 24. Media Player playing sample01.mp4](image)

4.2.2. Testing the software

Each time a request is made it is easy to check if the software works correctly and which is the final result. Because the CEL MPEG-21 contracts contain the conditions that rule the evaluation of the request, in this simulation the user can create allowed or not allowed request intentionally using the following table that contains all the correct combinations:

<table>
<thead>
<tr>
<th>User ID</th>
<th>Video to play</th>
<th>Delivery Mode</th>
<th>Broadcasting Mean</th>
<th>Country</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>ES111111111111</td>
<td>sample01.mp4</td>
<td>Linear</td>
<td>None</td>
<td>ES</td>
<td>cat</td>
</tr>
<tr>
<td>ES111111111111</td>
<td>sample01.mp4</td>
<td>NonLinear</td>
<td>None</td>
<td>ES</td>
<td>cat</td>
</tr>
<tr>
<td>ES111111111111</td>
<td>sample01.mp4</td>
<td>Linear</td>
<td>None</td>
<td>ES</td>
<td>esp</td>
</tr>
<tr>
<td>ES222222222222</td>
<td>sample02.mov</td>
<td>Broadcasting</td>
<td>Satellite</td>
<td>ES</td>
<td>esp</td>
</tr>
<tr>
<td>ES222222222222</td>
<td>sample02.mov</td>
<td>Broadcasting</td>
<td>Internet</td>
<td>ES</td>
<td>esp</td>
</tr>
<tr>
<td>EN333333333333</td>
<td>sample03.mp4</td>
<td>Linear</td>
<td>None</td>
<td>EN</td>
<td>eng</td>
</tr>
</tbody>
</table>

Table 2. All possible allowed request

There is a request that is inside the contract but it has expired already, this was done in order to do date checks:

<table>
<thead>
<tr>
<th>User ID</th>
<th>Video to play</th>
<th>Delivery Mode</th>
<th>Broadcasting Mean</th>
<th>Country</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN333333333333</td>
<td>sample03.mp4</td>
<td>NonLinear</td>
<td>None</td>
<td>EN</td>
<td>eng</td>
</tr>
</tbody>
</table>

Table 3. Request already expired

For a more detailed use of the request and the responses received, see Appendix 3 Detailed process of the software checking.
4.3. **Additional software characteristics**

4.3.1. **Programming language usage**

To develop this project, it has been needed the use of different programming languages which made its integration somehow more complex than the initial planning.

Excluding the CEL MPEG-21 contracts that are based in XML, the languages used are Python for all the API and the Media controller modules, Solidity to create the smart contract, JavaScript to develop the smart contract controllers and the looper script made in PowerShell. The percentages of the usage can be observed in the following plot:

![Figure 25. Programming language usage in the project](image)

As the Solidity and JavaScript are mandatory to interact with the Ethereum Blockchain, for future developments it could be interesting to expand the usage of JavaScript code because it will make the integration more solid and stable.

4.3.2. **Gas cost per transaction**

As Ethereum is a Blockchain environment each transaction that is made different from a call() function has a cost in the currency used, in this case Gas.

The Gas cost of each transaction is calculated based in the supposed computational cost that will require to implement, making the system more secure because it will avoid the storage of useless information and render spam unusable.

One important point to take into account is that the price of Gas is something variable and is determined by the Block Miners. If someone wants to check the current price of Gas it can consult (Ethereum, 2020).
To approximate the price of 1 storage data transaction that the software does for each request 32 different requests have been done and the Gas cost registered creating the following plot:

As it can be seen the average cost obtained is 65.779 Gas. Taken into account the currency value at 1th April 2021, the cost of 1 ETH is 1666,10€, and 1ETH equals to 120.435.627 Gas. From these values the following table can be done:

<table>
<thead>
<tr>
<th>Gas</th>
<th>ETH</th>
<th>Euros</th>
</tr>
</thead>
<tbody>
<tr>
<td>1th April currency</td>
<td>120.435.627</td>
<td>1</td>
</tr>
<tr>
<td>Transaction Average cost</td>
<td>65.779</td>
<td>0,0005461</td>
</tr>
</tbody>
</table>

The price of each request processed by the software will be (at the date of this document) about 0'91€.
4.4. **Goal’s checklist**

The main goals of this project are:

- To create a software utility that allows to extract the information contained in a digital contract in a format that can interact with other software.
- To develop a software capable of understanding and the extracted information and apply the statements inside the contract to discern if a request can be allowed or not.
- To relate somehow the information inside the digital contracts with another environment that has immense potential for future development: the smart contracts deployed in Blockchain.
- To implement a simulation that evaluates the statements in three different digital contracts and control if a user can display a multimedia content, creating an historic of the interactions with this content in the blockchain.

The following table shows the goals achieved checklist in this project:

<table>
<thead>
<tr>
<th>Main goal</th>
<th>Checklist</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extract</td>
<td>✓ API Parser</td>
<td>Development in Python of the API Parser that obtains the relevant information and returns it as a programming variable.</td>
</tr>
<tr>
<td></td>
<td>✓ Applying ontology statements</td>
<td>The internal workflow of the back end can determine if a request is allowed.</td>
</tr>
<tr>
<td></td>
<td>✓ Report creation</td>
<td>The integration between the JavaScript and the python part is done using a report file.</td>
</tr>
<tr>
<td>Software</td>
<td>✓ Media Agent Backend</td>
<td>Development in Python of a backend that can call the API, evaluate the data and discern the correct result.</td>
</tr>
<tr>
<td></td>
<td>✓ Media Agent Frontend</td>
<td>Development in Python of a frontend to process and inform the user of the results.</td>
</tr>
<tr>
<td></td>
<td>✓ Media Player</td>
<td>Integration of a Media Player to simulate the Act of the digital contracts.</td>
</tr>
<tr>
<td>Relate the information</td>
<td>✓ Solidity smart contract</td>
<td>Development of a smart contract that responds to the needs of the project.</td>
</tr>
<tr>
<td></td>
<td>✓ JavaScript controller</td>
<td>Development in JavaScript of a software that allows interaction with a deployed smart contract.</td>
</tr>
<tr>
<td>Simulation</td>
<td>✓ Development of all contracts</td>
<td>Development of CEL MPEG-21 contracts and a smart contract that are used in the simulation.</td>
</tr>
<tr>
<td></td>
<td>✓ Ethereum Blockchain Sandbox</td>
<td>Integration of the project with a Ethereum Sandbox to carry out the simulations</td>
</tr>
<tr>
<td></td>
<td>✓ Testing of the software</td>
<td>Testing of all the modules of the software in all the possible cases of the simulation.</td>
</tr>
<tr>
<td></td>
<td>✓ Evaluate the results</td>
<td>Evaluate the results of the simulation, if the goals were achieved and if the workflow is the intended.</td>
</tr>
</tbody>
</table>

*Table 5. Main goals accomplished checklist*
5. **Budget**

For the development of the budget, it has been considered two different types of costs: the direct, differentiating the personal and the equipment costs, and the indirect ones.

5.1. **Direct costs**

The direct costs are the expenditures made when the project was developed. The direct costs can be divided between personal and equipment. The direct costs add to **8.977,50€**. The following are a more detailed view of them.

5.1.1. **Costs of personal**

The costs of personal mainly consists in the salary of a junior engineer that develops the project. In the project it will be stated that only one junior engineer has worked in it.

The estimation of hours worked in the project is stated at 12 hours per week. As it can be seen in the Gantt Diagram in the point 1.5 the project consisted in a total of 36 weeks.

Supposing that an average salary of a junior engineer is **20,00€/hour** then the total personal costs of the project is about **8.640,00€**.

5.1.2. **Costs of equipment**

The costs of equipment are referred to the objects needed in the development of the project. Also, all the license used in the development of the project will be included in this section. Because some objects and licenses can be used for a long time even after the completion of the project an amortization has been applied.

The summary of the equipment costs can be seen in the following table:

<table>
<thead>
<tr>
<th>Object</th>
<th>Price</th>
<th>Lyfecycle</th>
<th>Cost per year</th>
<th>Used time</th>
<th>COST</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC</td>
<td>1.500,00€</td>
<td>3 years</td>
<td>500,00€</td>
<td>6 months</td>
<td>225,00€</td>
</tr>
<tr>
<td>Python License</td>
<td>0,00€</td>
<td>10 years</td>
<td>0,00€</td>
<td>6 months</td>
<td>0,00€</td>
</tr>
<tr>
<td>Solidity License</td>
<td>0,00€</td>
<td>10 years</td>
<td>0,00€</td>
<td>6 months</td>
<td>0,00€</td>
</tr>
<tr>
<td>JavaScript License</td>
<td>0,00€</td>
<td>10 years</td>
<td>0,00€</td>
<td>6 months</td>
<td>0,00€</td>
</tr>
<tr>
<td>Truffle Suite (truffle and Ganache) License</td>
<td>0,00€</td>
<td>5 years</td>
<td>0,00€</td>
<td>6 months</td>
<td>0,00€</td>
</tr>
<tr>
<td>ISO/IEC 21000-20:2016 (CEL MPEG-21 Standard)</td>
<td>225,00€</td>
<td>1 year</td>
<td>225,00€</td>
<td>6 months</td>
<td>112,50€</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>337,50€</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6. Costs of equipment

5.2. **Indirect costs**

The indirect costs are all the expenses derived from heating, electricity, internet price and other similar services. Because there is not enough information to accurate define the budget inverted in this section, the indirect costs will be calculated assuming that they are 20% of the direct ones.

Therefore, the indirect costs are 20% of 8.977,50€ which results in **1.795,50€**.
5.3. **Budget summary**

The total cost of the project can be seen in the following table:

<table>
<thead>
<tr>
<th>Type</th>
<th>COST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct costs</td>
<td></td>
</tr>
<tr>
<td>Personal</td>
<td>8.640,00€</td>
</tr>
<tr>
<td>Equipment</td>
<td>337,50€</td>
</tr>
<tr>
<td>SUBTOTAL</td>
<td>8.977,50€</td>
</tr>
<tr>
<td>Indirect costs</td>
<td></td>
</tr>
<tr>
<td>SUBTOTAL</td>
<td>1.795,50€</td>
</tr>
<tr>
<td>TOTAL</td>
<td><strong>10.773,00€</strong></td>
</tr>
</tbody>
</table>

Table 7: Total costs of the projects

The **total cost of the project** is the sum of both of them which adds to **10.773,00€**.
6. **Environment Impact**

During the realization of the project, the only environment impact done was the energy consumed by the computer while during the development. This energy is of an unknown origin, it can be provided from green sources like wind turbines or hydroelectric plants or pollutant ones such as fuel or nuclear plants.

The main point to take into account is the use of Blockchain. During the development a sandbox Blockchain with all the blocks mined as wanted was used, so it had no impact. In future developments, there are 2 options: use a private Blockchain with no crypto currency that has no need to mine the blocks and has a negligible effect on the environment, or deploy the smart contract in the real Ethereum Blockchain, which will have a considerable environmental impact.

Ethereum has a large environmental impact caused by the massive amount of energy needed to mine each one of the blocks. The carbon footprint of a single transaction is calculated to be about 468,04 kgCO2 and only about 40% of the energy used to mine the Blockchain is created using green sources.

The fact that the software will do one transaction for each user request has a negative impact on the environment as it will increment the need to mine new blocks in the future.
7. **Conclusions and future development**

7.1. **Conclusions**

The development of the CEL MPEG-21 standard has been one step forward in the digitalization of contracts regarding Multimedia Content, which have always been classical text contracts. This opened a lot of new possibilities to interact with them and try to automate the previous processes that used the classical contracts.

One of these possibilities are the smart contracts deployed in a Blockchain. These smart contracts are already used to substitute some intermediary agents as they can be programmed to act depending on the inputs received.

These two kinds of contracts can be used for the companies to be faster and more efficient when dealing with multimedia content controlled by contracts. Because the Blockchain is a decentralized and transparent environment, it is possible to keep the records of the transactions done with the multimedia content through a smart contract.

In this project, the main goal has been to develop this relationship between the two contracts and also use to the fullest the fact that the statements regulating a Media object are in a digital format.

During the development of the software, it has been determined that a direct translation of a CEL MPEG-21 to a smart contract does not seem useful and easy to implement. However, another relationship between the contracts has been developed.

To maximize the utility of the digital statements contained in the CEL MPEG-21 contracts, an agent that can control what can be allowed and what cannot over a Media content has been created. This agent works as an intermediary between the parts of the classical contracts and fulfils the purpose of automating previous manual processes.

Also, the manager of a smart contract deployed in the Blockchain has been developed, which is related to the agent and keeps track of all the interactions done with a Media content. This functionality has the aim to improve the tracking of the use of a Media object.

In the end, the main goal of the project has been achieved, developing software that is functional and relates two independent fields like the CEL MPEG-21 standard contracts and the Blockchain smart contracts.
7.2. **Future developments**

One of the main issues about using Blockchain is that doing a transaction may cost crypto money. This characteristic is intrinsic of the Ethereum Blockchain and cannot be changed, so, it would be useful to study other Blockchain options that could be more attractive to be used in the economic aspect.

Another point to expand the software of the project could be trying to change the asynchrony of the communication between the Media Agent and the manager of the smart contract. As it has been detailed in the point 4.3 the 80% of the project is done in python. Because the use of JavaScript is needed to implement the manager of the smart contract the percentage of JavaScript cannot be reduced. Then, the other option would be implementing some actual Python parts in JavaScript, which can change the way the agent communicates with the smart contract.

Furthermore, the smart contracts in Blockchain can be expanded and new functionalities included. One of the points that this project has not included in its scope is the remuneration part stated in the CEL MPEG-21 pane section explained in the point 2.1. If all Parties included in the contract have associated a Ethereum hash user each one then it is also possible to include in the smart contract the money transactions between them stated in the contract using for the payment crypto money.

In this way it will be automated all the economic section of a contract, not only the requirements for allowing an Act.
Bibliography


Appendices

Appendix 1. CEL MPEG-21 Standard detail

The following is a detailed explanation of all variables in the CEL MPEG-21 standard and the uses of each one.

Appendix 1.1. Core Section

The upper section in the Hierarchy of a contract is expressed only by core variables. This upper level can be seen in this diagram:

The most important variables are the following:

- **contractId**: the contract identifier.
- **governingLaw**: The country or Law system applicable.
- **court**: indicates which court has jurisdiction over the terms in the contract.
- **isCourtJurisdictionExclusive**: Boolean that defines if the court is agreed to be exclusive.
- **Metadata**: Element that allows to introduce metadata related to the contract using Dublin Core placeholders.
- **ContractsRelated**: a relational variable that refers to other identified contracts and express which relation they have with the current contract (supersedes, cancels, prevailsOver or isAmendOf).

- **Party**: this element defines all the parties for which the contract is binding. The parties can be of 2 different types that share the same structure: Person and Organization. In both of them some basic information such as Name, identifier, description etc can be found besides the Signature of the party.

- **Body**: the body contains almost all the information of the contract. It is such a large element that a more exhaustive description is needed to understand this element.

**Body element**

The first attribute of the body element is the `textualPart`. In this attribute it can be defined a lot of `TextClauses` which will be referred later in the operative part so an identifier is needed for each one of them.

The second and main attribute of the body is the `OperativePart`. Inside this attribute all the Deontic Structure is found, containing all the clauses that state Permission, obligation, prohibition or statement. The `DeonticStructureBlock` is an aggregation of other Permissions, Obligations, Prohibitions or Statements.

![Figure 28. CEL MPEG-21 operative part structure](image-url)
At the end, all the deontic clauses have the same structure type:

Figure 29. CEL MPEG-21 deontic clause structure

The main ones being:

- **Number**: number of the deontic structure clause.
- **Id**: identifier of the clause.
- **ideRef**: all the references to the textCauses in the textual part.
- **Metadata**: additional information about the clause itself.
- **Context**: information of any type added to the clause.
- **Precondition**: a condition that should be true before the the associated act is performed.
- **Subject**: a reference to a party already defined in the cel-core:Party section.
- **Act**: specifies the right that is permitted/obligated/prohibited to the cel-core:Subject if the constraints, pre-conditions and post-conditions (if present) are met.
- **Object**: the resource against which the deontic expression will apply.
- **ResultantObject**: represents the new resource resultant to the application of the right in the Act to the Object.
- **Constraint**: simple conditions, restrictions and contraints that help to define the deontic clause.
− **PostCondition**: a condition that needs to be true after the Act is finished.
− **Issuer**: a reference to the cel-core:Party element already defined that issues the permission, prohibition, obligation or statement specified in the deontic clause.

### Appendix 1.2. Ipre Section

The elements defined inside the ipre section are related to 3 complex elements in the core section that need a more exhaustive way to detail their characteristics. These core elements are: cel-core:Act, cel-core:Constraint and cel-core:PermissionAttributes.

#### Act element

Inside the act element there can be some more detailed ipre elements based in the cel-ipse:ExploitIPRights which is the root of all the action of exploiting the intellectual property. Such acts can be:

− Distribute: sell, rent and lent audiovisual material.
− Duplicate: to make copies of the work.
− Fixate: which represents the action of recording a performance.
− PublicComunication: represents the act of publicly performing.
− Transform: for all modification.

#### Constraint element

The element Constraint requires modelling conditions for the audiovisuals contracts. These are all based in the ipre element ExploitationCondition and the children based in that:

− **AccessPolicy**: specifies if the user has to pay or not for the content fruition.
− **CopyrightExceptionFact**: specifies copyright exceptions.
− **DeliveryModality**: restricts the communication to the public.
− **Device**: restricts the channel or type of equipment used.
− **IPEntityContext**: restricts a cel-core:Object to be part of another object identified.
− **Language**: restricts the language permitted/obligated/prohibited for public communication.
− **Length**: restricts the maximum duration.
− **MaterialFormat**: restricts the technical properties of the material.
− **Means**: restricts the format in which the content is delivered to the public.
− **Runs**: restricts the maximum number of executions of the action.
− **ServiceAccessPolicy**: restricts the access to the services.
− **ServiceChannelContext**: restricts the channel used to for the exploitation of the rights.
- **SpatialContext**: restricts the territory on which the action can be executed.
- **TemporalContext**: restricts the time period of execution of the action.
- **UserTimeAccess**: restricts the time availability of the content.

**Permission Attributes**

Some cel-core:Permission elements require specific values when dealing with intellectual property rights. The main information added in that case is:

- **Percentage**: when the permission is shared with more than 1 actor.
- **incomePercentage**: when the income resulting of the exploitation is shared.
- **isExclusive**: indicates if the exploitation may be granted by the issuer to multiple licensees.
- ** sublicenseRight**: indicates if it is possible to sublicense a granted right.

**Appendix 1.3. Pane Section**

The elements in the pane section are used to specify information about payments and notifications. As such there are 2 main elements types: Payment and Notify.

**Payment**

The payment element can be found inside of the cel-core:Act element. It models a payment action between the contracted parties. It also has to be specified in an Act inside of a cel-core:Obligation element. The main attributes that define a Payment are:

- **Amount**: the amount to be paid.
- **Currency**: the currency in which the amount will be paid.
- **incomePercentage**: it attributes the payment when it is defined as a percentage of the income from the action.
- **Beneficiary**: a party reference to the entity that will receive the payment.
- **IncomeSource**: when the income comes from a percentage of the income from the action here it is referred that Act.

**Notify**

The element Notify is found inside the cel-core:Act element. It models a notification of an Act to a specific party. As such, it only contains 2 attributes

- **Recipient**: a reference to the Party that is needed to notify.
- **About**: a reference to the Act that has to be notified.
Appendix 2. Configure Blockchain environment

The project is conceived with the premise that when a simulation starts the Solidity smart contract is already deployed in the Ethereum Blockchain. For this reason, if the simulation is the first one done in the local terminal a previous environment configuration is needed.

The Ethereum Blockchain will be used the last part of the software working. As every transaction has a cost in Gas and you may need to have a real account to use the real Blockchain, in the simulation it will be used a sandbox of Ethereum Blockchain installed locally. This tool is part of the Truffle Suite and is called Ganache, for more information see (ConsenSys Software, 2021).

To configure the local blockchain the follow parameters have to be indicated: Gas cost of a block, gas limit of a transaction, network, port and mining status. Once it has started the Blockchain you can see them like this:

![Ganache configuration parameters](image)

Figure 30. Ganache configuration parameters

At this point you can deploy your Solidity smart contract using Truffle. As it has to be compiled first, the code used shall be:

```
C:\Users\Carles\Documents\TFG\Codi\Codi>truffle compile
Compiling your contracts...
> Compiling :\contracts\Storage.sol
> Artifacts written to C:\Users\Carles\Documents\TFG\Codi\Codi\build\contracts
> Compiled successfully using:
  - solc: 0.7.0+commit.9e61f92b.Emscripten clang
```

Figure 31. Truffle compiling smart contract
Later, it can be deployed like this:

![Later, it can be deployed like this:](image)

**Figure 32. Truffle deploying smart contract**

It is important to observe if the port used by Ganache is the same as the one indicated in the network inside the `truffle-config.js` file:

```javascript
networks: {
  development: {
    host: "127.0.0.1", // Localhost (default: none)
    port: 7545, // Standard Ethereum port (default: none)
    network_id: "*", // Any network (default: none)
  },
}
```

**Figure 33. Blockchain network parameters code**

At this point the smart contract is deployed.

![At this point the smart contract is deployed.](image)

**Figure 34. Ganache contract deployed**
Appendix 3. Detailed process of the software checking

To test the behaviour of the software the following types of request will be made and analysed:

- An allowed request.
- One with a wrong User ID.
- One with a video requested that the user have no permissions.
- One with an erroneous delivery mode.
- One broadcasting with a wrong mean.
- One with a wrong country.
- One with a language that does not apply.

These requests will englobe all the possible obtainable results from the backend and it will observe the behaviour when storing each one of them in the blockchain.

Appendix 3.1. Checking Allowed request

This request will check the behaviour of the software when it has to process a user petition that complies with the statements of one of the CEL MPEG-21 contracts.

1. The request will consist in the following data:

<table>
<thead>
<tr>
<th>User ID</th>
<th>Video to play</th>
<th>Delivery Mode</th>
<th>Broadcasting Mean</th>
<th>Country</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>ES2222222222</td>
<td>sample02.mov</td>
<td>Broadcasting</td>
<td>Satellite</td>
<td>ES</td>
<td>esp</td>
</tr>
</tbody>
</table>

Table 8. Allowed request to check

2. When executing the software and passing this request it opens the Media player and the sample02.mov is the only file inside the public video folder:
3. The report.txt contains the following information:

```plaintext
urn:VATIN:ES2222222222
sample02.mov
Broadcasting
2021-03-30T13:25:09.468452
ES-esp
APPROVED
```

*Figure 36. Resultant report from allowed request*

4. After a few seconds the looper detects the information inside the report.txt and triggers the web3_setter.js to store it in the deployed smart contract:

```plaintext
ADDRESS
0x8F541Be537A4820E86F5baa410fd4E7aC11aff37

CREATION TX
0x68459946c4ea2A1a66523589F970F079E120a4626c401fb8dE5033353bf387c
```

*Figure 37. Ganache storage post allowed request check*

The conclusion is that the software behave exactly as it was intended and the loading of the data in the Blockchain is done perfectly.
Appendix 3.2. Wrong user request

This request will check if the software is able to identify if a user is part of a contract or not.

1. The request will be the same as the last one with the variation of the user ID:

<table>
<thead>
<tr>
<th>User ID</th>
<th>Video to play</th>
<th>Delivery Mode</th>
<th>Broadcasting Mean</th>
<th>Country</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>ES44444444444</td>
<td>sample02.mov</td>
<td>Broadcasting</td>
<td>Satellite</td>
<td>ES</td>
<td>esp</td>
</tr>
</tbody>
</table>

Table 9. Wrong user request to check

2. When running the software and introducing this request the following window appears:

![User not in any contract](image)

Figure 38. Results window from check request 2

3. The report.txt contains the following information:

```plaintext
urn:VATIN:ES44444444444
sample02.mov
Broadcasting
2021-03-31#13:25:38.488452
ES-esp
REJECTED: User not in any contract
```

Figure 39. Resultant report from wrong user request

4. The data stored at the end in the blockchain is:

![Ganache storage post wrong user request check](image)

Figure 40. Ganache storage post wrong user request check
Appendix 3.3. Wrong video request

This request will check if the software is able to identify if a user has a contract with the requested video or not.

1. The request will be the same as the allowed one with the variation of the Video to play:

<table>
<thead>
<tr>
<th>User ID</th>
<th>Video to play</th>
<th>Delivery Mode</th>
<th>Broadcasting Mean</th>
<th>Country</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>ES22222222222</td>
<td>sample01.mp4</td>
<td>Broadcasting</td>
<td>Satellite</td>
<td>ES</td>
<td>esp</td>
</tr>
</tbody>
</table>

*Table 10. Wrong video request to check*

2. When running the software and introducing this request the following window appears:

![Result](image)

*Figure 41. Results window from check request 3*

3. The report.txt contains the following information:

```
urn:VATIN:ES22222222222
sample01.mp4
Broadcasting
2021-03-31#13:26:01.168452
ES-esp
REJECTED: Video not related to user
```

*Figure 42. Resultant report from wrong video request*

4. The data stored at the end in the blockchain is:
Appendix 3.4. Wrong delivery method request

This request will check if the software is able to identify if a user that has a contract with the requested video have permission with the delivery method or not.

1. The request will be the same as the allowed one with the variation of the delivery mode:

<table>
<thead>
<tr>
<th>User ID</th>
<th>Video to play</th>
<th>Delivery Mode</th>
<th>Broadcasting Mean</th>
<th>Country</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>ES2222222222</td>
<td>sample02.mov</td>
<td>Linear</td>
<td>Satellite</td>
<td>ES</td>
<td>esp</td>
</tr>
</tbody>
</table>

   Table 11. Wrong delivery method request to check

2. When running the software and introducing this request the following window appears:

   ![Error delivery method declared not allowed](image)

3. The report.txt contains the following information:

   ```
   urn:VATIN:ES22222222222
   sample02.mov
   Linear
   2021-03-31#13:27:11.168452
   ES-esp
   REJECTED: delivery method declared not allowed
   ```

   Figure 45. Resultant report from wrong delivery method request
4. The data stored at the end in the blockchain is:

```
{ 6 items
  userID : string "urn:VATIN:ES22222222222..."
  videoID : string "sample02.mov"
  modality : string "Linear"
  dateRequest : string "2021-03-31"
  countryLang : string "ES-esp"
  result : string "REJECTED: delivery m..."
}
```

Figure 46. Ganache storage post wrong delivery method request check

Appendix 3.5. Wrong mean request

This request will check if the software is able to identify if a user that has a contract with the requested video have a permission where the delivery method is Broadcasting and can identify the permission Mean or not.

1. The request will be the same as the allowed one with the variation of the delivery mode:

<table>
<thead>
<tr>
<th>User ID</th>
<th>Video to play</th>
<th>Delivery Mode</th>
<th>Broadcasting Mean</th>
<th>Country</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>ES222222222222</td>
<td>sample02.mov</td>
<td>Broadcasting</td>
<td>None</td>
<td>ES</td>
<td>esp</td>
</tr>
</tbody>
</table>

Table 12. Wrong mean request to check

2. When running the software and introducing this request the following window appears:

```
Result

Error Broadcasting Mean declared not allowed
OK
```

Figure 47. Results window from check request 5

3. The report.txt contains the following information:

```
urn:VATIN:ES22222222222
sample02.mov
Broadcasting-None
2021-03-31#13:34:01.008452
ES-esp
REJECTED: Broadcasting Mean declared not allowed
```
4. The data stored at the end in the blockchain is:

```json
STORAGE

{
  userID : string "urn:VATIN:EN33333333333",
  videoID : string "sample03.mp4",
  modality : string "Broadcasting-None",
  dateRequest : string "2021-03-31",
  countryLang : string "en-eng",
  result : string "REJECTED: The request is outdated"
}
```

Figure 49. Ganache storage post wrong mean request check

Appendix 3.6. Wrong date request

This request will check if the software is able to identify if a request is inside the date range stated in the contract.

1. The request will be the one that has been designed with a valid date range already expired:

<table>
<thead>
<tr>
<th>User ID</th>
<th>Video to play</th>
<th>Delivery Mode</th>
<th>Broadcasting Mean</th>
<th>Country</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN33333333333</td>
<td>sample03.mp4</td>
<td>NonLinear</td>
<td>None</td>
<td>EN</td>
<td>eng</td>
</tr>
</tbody>
</table>

Table 13. Wrong date request to check

2. When running the software and introducing this request the following window appears:

   ![Result window from check request 6](image)

   Figure 50. Results window from check request 6

3. The report.txt contains the following information:

   ```
   urn:VATIN:EN33333333333
   sample03.mp4
   NonLinear
   2021-03-31#13:41:28.934652
   EN-eng
   REJECTED: The request is outdated
   ```
Figure 51. Resultant report from wrong date request

4. The data stored at the end in the blockchain is:

```
ADDRESS
0x8f541be537a4820f86f5baa410fd4e7ac11aff37

CREATION TX
0x080459946c4ea21a6523589f976f076e120a4626c401f8de50d33353bf307c
```

**STORAGE**

```
{ 6 items
    userID: string "urn:VATIN:ES333333333333333",
    videoID: string "sample02.mp4",
    modality: string "NonLinear",
    dateRequest: string "2021-03-31",
    countryLang: string "EN-eng",
    result: string "REJECTED: The request...
}
```

Figure 52. Ganache storage post wrong date request check

**Appendix 3.7. Wrong country request**

This request will check if the software is able to identify if the country from the request matches the one stated in the contract.

1. The request will be the same as the allowed one with the variation of the country:

<table>
<thead>
<tr>
<th>User ID</th>
<th>Video to play</th>
<th>Delivery Mode</th>
<th>Broadcasting Mean</th>
<th>Country</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>ES222222222222</td>
<td>sample02.mov</td>
<td>Broadcasting</td>
<td>Satellite</td>
<td>EN</td>
<td>esp</td>
</tr>
</tbody>
</table>

Table 14. Wrong country request to check

2. When running the software and introducing this request the following window appears:

![Image of error message](image)

Figure 53. Results window from check request 7

3. The report.txt contains the following information:

```
urn:VATIN:ES22222222222
sample02.mov
Broadcasting
2021-03-31#13:55:04.984252
EN-esp
REJECTED: country declared not allowed
```

Figure 54. Resultant report from wrong country request
4. The data stored at the end in the blockchain is:

```
Figure 55. Ganache storage post wrong country request check
```

**Appendix 3.8. Wrong language request**

This request will check if the software is able to identify if the language from the request matches the one stated in the contract.

1. The request will be the same as the allowed one with the variation of the language:

<table>
<thead>
<tr>
<th>User ID</th>
<th>Video to play</th>
<th>Delivery Mode</th>
<th>Broadcasting Mean</th>
<th>Country</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>ES2222222222</td>
<td>sample02.mov</td>
<td>Broadcasting</td>
<td>Satellite</td>
<td>ES</td>
<td>cat</td>
</tr>
</tbody>
</table>

*Table 15. Wrong language request to check*

2. When running the software and introducing this request the following window appears:

```
Figure 56. Results window from check request 8
```

3. The report.txt contains the following information:

```
Figure 57. Resultant report from wrong language request
```
4. The data stored at the end in the blockchain is:

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
Appendix 4. Code
The code of the software developed in this project can be found in the ZIP file attached to the project delivery. It can also be found in this public GIT project page: https://github.com/carlesbertran/Codi
Glossary

ABI: Application Binary Interface
API: Application Programming Interface
CEL: Contract Expression Language
ETH: Ether
ID: Identifier
IEC: International Electrotechnical Commission
IMT: Institute Mines-Télécom
ISO: International Organization for Standardization
JSON: JavaScript Object Notation
MPEG: Moving Picture Experts Group
OS: Operating System
RPC: Remote Procedure Call
TSP: Télécom SudParis
WG: Work Group
XML: Extensible Markup Language