www.itcon.org - Journal of Information Technology in Construction - ISSN 1874-4753

OPEN-SOURCE COMPLIANCE CHECK WEB APPLICATION FOR DIGITAL BUILDING PERMITS BASED ON A VISUAL PROGRAMMING LANGUAGE

SUBMITTED: April 2026
PUBLISHED: November 2026
EDITOR: Robert Amor
DOI: 10.36680/j.itcon.2026.062

Bruno Muniz, PhD student
University of Minho, ISISE, ARISE, Department of Civil Engineering, Guimarães, Portugal
https://orcid.org/0000-0002-3183-9779
id11378@alunos.uminho.pt

Jose Granja, Assistant Reseacher
University of Minho, ISISE, ARISE, Department of Civil Engineering, Guimarães, Portugal
https://orcid.org/0000-0002-0858-4990
granja@civil.uminho.pt

Miguel Azenha, Associate Professor
University of Minho, ISISE, ARISE, Department of Civil Engineering, Guimarães, Portugal
https://orcid.org/0000-0003-1374-9427
miguel.azenha@civil.uminho.pt

SUMMARY: *The automation of compliance checks for building permits is still challenging, due to the extensive regulatory frameworks and complex methods to implement machine-readable regulations. Some works already propose visual programming languages as a way to simplify the creation of machine-readable regulations, potentially enabling its creation by municipal officers. Yet, these studies either have a conceptual approach or a proprietary implementation. This work presents an end-to-end approach to automate building permit compliance check with a web-application prototype. It includes a visual programming language, allowing users to create custom permit rules using local terminology, and a domain-specific model that translates local terms into the IFC (Industry Foundation Classes) standard. The architecture proposed uses open-source tools and open standards which facilitates its inspection and adaptability to other scenarios. The scope of this study is the Portuguese building permit context: the domain model, the rules and the use case are based on Portuguese regulations. To demonstrate the rule creation and compliance check capabilities, nine representative clauses from Portuguese national and municipal regulations are implemented as visual rules, three of which are presented here, and their execution on an IFC model is compared with independent measurement.*

KEYWORDS: *digital building permits, automatic compliance checking, BIM, IFC, visual programming language.*

REFERENCE: *Muniz, B., Granja, J., & Azenha, M. (2026). Open-source compliance check web application for digital building permits based on a visual programming language. Journal of Information Technology in Construction (ITcon), 31, 1558-1586. https://doi.org/10.36680/j.itcon.2026.062*

COPYRIGHT: *© 2026 The author(s). This is an open access article distributed under the terms of the Creative Commons Attribution 4.0 International (https://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.*

1. INTRODUCTION

The building control system ensure that parameters such as safety, comfort, performance and sustainability are addressed in the development of construction works, with the final issuance of a permit (Pedro et al., 2011). In other words, the quality of the building stock is directly connected to the capability of the state to guarantee an efficient building permit process and to establish a coherent and clear set of rules to be applied. In Portugal, as in many other European countries, the process to issue a permit occurs at a municipality level (Fauth et al., 2025), where, among many activities, the compliance of the building plans is verified by technicians against a series of requirements comprised in a regulatory framework. Despite its importance, the process is still mainly manual-based, supported by CAD or PDF documents, being time and cost-inefficient, lacking transparency, and being error-prone (Noardo et al., 2022).

The automation of the compliance checks is a largely researched subject with several published studies in the past decades (Amor & Dimyadi, 2021). The interpretation of regulations is an elemental activity in the automation of compliance check (Eastman et al., 2009). It involves the translation of the natural language of its documents into a logical structure, and its implementation using machine-interpretable language. However, this activity is one of the most challenging, due to factors such as the extensiveness of the regulatory frameworks, the subjectivity and ambiguity found in the regulations and the domain concepts implemented in its text, that often represent local urbanism definitions that can change in different cities of the same country (Soliman-Junior et al., 2021). This scenario imposes multidisciplinary teams to automate compliance checks, including programming experts, Building Information Modelling (BIM) experts, and local permit experts, who are specialised in regulations and their concepts, and therefore can mitigate the gaps and possible misunderstandings in this process.

Currently, several tools enable the execution of compliance checks on BIM models, but either they are implemented with hard-coded rules that can be applied in a specific local permit context, or they lack flexibility to elaborate new rules (Nawari, 2019). Therefore, there is a need to be capable of implementing local permit concepts and yet grant flexibility to make the tool scalable. The scalability can also be granted by empowering the local permit experts to create machine-readable regulations. This aligns with the efforts that present Domain Specific Languages (DSLs) to allow domain experts to create programming code, since they incorporate domain concepts that can be familiar and shorten the adoption barrier for these professionals (Lee, 2011). Still, most of the DSLs can be difficult to use by permit experts, since these languages are visually similar to general-purpose programming and query languages, such as Python and SQL. Another approach within the DSLs is to use Visual Programming Languages (VPL) to shorten even more the adoption barrier (Preidel et al., 2017), yet these approaches are generally focused on design, general compliance check needs, or adopting local concepts that are not replicable for other scenarios.

This paper proposes an open-source approach to the creation of rules and the execution of compliance checks for building permits on an IFC model and demonstrates how the lower level of abstraction from programming languages and the IFC schema can be bridged to a higher level on a VPL. Accordingly, the research question guiding this study is: How can the lower level of abstraction of programming languages and the IFC schema be bridged to a higher level in a VPL that incorporates local building permit terminology? As a result, an end-to-end design and development of the open-source web application LiMA Blocks is presented (source code available at https://github.com/Muniz1994/LimaBlocks), which allows the creation of machine-readable regulations and the execution of compliance checks on IFC models. It incorporates a VPL focused on building permit compliance checks and an architecture based on modular components that can be adapted for different local permit needs. Its application is demonstrated through a use case in which municipal clauses from Vila Nova de Gaia and Lisbon and a national clause are implemented as visual rules and checked on an IFC model of a mixed-use building. This application does not aim to provide an overall solution to automate compliance checking within building permits, but rather to promote an innovative solution that facilitates the automation of rules for building permits. The regulations used as the basis for this work come from the Portuguese regulatory framework, due to the history of active collaboration of the authors with Portuguese municipalities. Therefore, the scope of this study is the Portuguese building permit context, and the transfer of the methods to other regulatory frameworks is expected from its modular design and not a result demonstrated here.

The remaining sections are organised as follows. Section 2 presents the related work, covering the translation of regulatory text into machine-readable rules, the text-based and visual-based domain specific languages developed

CC BY

for BIM, and a comparative analysis of the compliance checking systems currently available in practice. Section 3 describes the methodology, setting out the overall considerations that guided the work and the specific methods adopted for the domain model, for the VPL and for the architecture of the application, as well as the approach and the scope of the evaluation. Section 4 presents the design of the application, namely the domain model that maps local permit concepts to the IFC data model, the visual programming language used to author the rules, and the system architecture. Section 5 reports the use case, following the creation of a rule from the Portuguese regulatory framework, the preparation of the model and the execution of the code and the resulting report, for a building modelled on actual allotment plans for Vila Nova de Gaia. Section 6 discusses the results, the benefits and the limitations of the proposed approach. Finally, Section 7 presents the conclusions.

2. RELATED WORK

Compliance check of building information serves as a pivotal task in the process to issue a building permit, ensuring that the structure adheres to the fundamental standards for design and safety. This section explores the necessary technical needs for extracting rules from documents and transforming them into machine-readable rules, as well as the DSLs that can be used as a flexible and scalable solution to implement machine-readable rules for a specific domain and finally the main software capable of executing compliance checks on BIM models.

2.1 From regulatory text to machine-readable rules

As defined by Eastman et al. (2009), rule checking can be structured into four stages: interpreting natural language regulations into machine-readable format, preparing the model with necessary information, executing rules, and generating reports. The first stage, rule interpretation, involves transforming human-readable rules into a format that machines can understand (Zhang & El-Gohary, 2017), and it imposes a major challenge due to the complexity and subjectivity of regulatory documents, which can include text, tables, and equations (Noardo et al., 2022).

Traditional interpretation methods directly translate (i.e. hardcoding) regulatory text into programming languages, requiring multidisciplinary teams and often leading to issues like limited scalability, reduced transparency, and high maintenance costs (Nawari, 2019). To address these, various methods have emerged over the past 60 years. Tabular methods, presented by Fenves's decision tables and the provision level of the SASE model, were among the early attempts to represent codes in a computable form (S. Fenves, 1966; S. J. Fenves et al., 1987) . This method structures rules in a matrix of decisions and criteria (Tan et al., 2010). Similarly, parametric tables are used in systems like Solibri[1]. Despite its simplicity, the implementation of tabular methods also involves a high level of hardcoding, suffering from the same scalability problems as the direct translation methods. In proprietary tools, the hardcoded and closed-source code also generates “black-boxes”, where the methods used to calculate permit parameters are not inspectable, which is detrimental to the transparency needed for building permit processes (Zijing et al., 2023).

Other approaches focus on interpreting natural language through manual markup or automated processes. The Requirement, Applicability, Selection, Exception (RASE) method uses constructs like requirements and exceptions to organise rule elements (Hjelseth & Nisbet, 2011). This method allows extracting, manually or assisted by software, the main components of natural language rules, enabling their systematisation and possible implementation through an extensive mapping between the entities extracted and entities from a formal data model such as IFC. Natural Language Processing (NLP) methods often aim to automatically interpret regulations. They can use rule-based and machine learning approaches to process the natural language rules, being also incorporated with semantic techniques to minimise errors from interpretation (Zhang & El-Gohary, 2017). However, these methods still carry drawbacks with the difficulties of dealing with ambiguities and vagueness of rules as well as the incapability of being 100% accurate to create the machine-readable format and map to models, which implies the remaining need for multidisciplinary teams.

An initiative denominated "Rules as Code" (RaC) aims to turn law into machine-readable formats (Waddington, 2020). Originating in New Zealand and expanding globally, RaC uses technology in policy development to reduce ambiguity. While not always producing executable code, RaC implementations often utilise markup languages like

[1] https://help.solibri.com/hc/en-us/articles/1500004623681-Rule-Parameters

CC BY

RuleML (Arnold, 2022) and Akoma Ntoso (Palmirani & Vitali, 2011), and VPLs such as Blawx (Morris, 2020). Despite advancements in RaC, concerns persist regarding technological readiness and potential errors due to disconnects between coders and legal professionals (Witt et al., 2024). The RaC initiative still lacks a consensus standard, and despite the relevance of formats such as LegalRuleML[2], the implementation of rule checking systems might be more adequate with more robust languages and systems, as the case of Blawx, which used the Flora-2 system (Yang et al., 2003).

2.2 Domain specific languages

Domain-Specific Languages bridge the gap between building permit experts and machine-readable code. These languages leverage high-level abstraction and specialised terminology, making them accessible to domain experts. Numerous DSLs exist within the construction sector, each tailored to specific use cases within the construction value chain, reflected in their specialised vocabulary (functions, objects, elements).

DSLs for the construction sector can be split into two main approaches:

a) Scripting DSLs: These use textual syntax, similar to general-purpose programming languages.
b) Visual DSLs: These employ visual symbols for coding, making them more accessible to non-programmers.

The following subsections present the most significant scripting and visual DSLs relevant to the construction sector.

2.2.1 Scripting languages

The primary goal of DSLs for BIM is to automate data processing and assessment of BIM models, though not exclusively for building permits.

Early efforts date back to Eastman and Henrion's 1977 GLIDE language, designed to aid CAD application development using object-oriented principles (Eastman & Henrion, 1977). Later, in 2011, the Building Environment Rule and Analysis (BERA) Language emerged for defining and verifying rules from BIM models. BERA's core, the BERA Object Model (BOM), abstractly represents building elements and can link to BIM models like IFC (Lee, 2011). While initially focused on spatial elements for circulation and programming, its reliance on the proprietary Solibri Model Checker (SMC) for IFC processing poses a significant limitation.

Subsequently, Mazairac & Beetz (2013) introduced BIMQL, a query language for IFC data extraction, deeply integrated with the BimServer.org platform. The BIM Rule Language (BIMRL) aimed to standardise building rule checking, operating similarly to SQL by extracting and querying IFC data stored in a database (Dimyadi, 2016). KBimCode was developed to translate the Korean Building Act into a machine-readable format, focusing on converting natural language rules rather than direct compliance checks. It can transform into Python for use in software like KBimAssess for IFC model compliance (Park & Lee, 2016). More recently, Daum and Borrmann's QL4BIM provides both scripting and visual interfaces to simplify BIM model querying for domain experts. QL4BIM prioritises user-friendliness by abstracting low-level complexities and uses simplified syntax, though users still need IFC data model knowledge (Daum & Borrmann, 2015). Its notable feature is step-by-step code execution with progressive, even 3D, result displays.

2.2.2 Visual programming languages

While scripting DSLs aim to simplify programming for domain experts, their syntax often remains complex. To address this, VPLs, which use visual elements like nodes, lines, and boxes, offer a more intuitive approach to depicting programming logic. This method is well-established in construction, particularly with proprietary tools like Revit and Rhino, though its application to IFC model processing is still evolving. Early efforts include the visual graph method for encoding design knowledge and aiding structural design by allowing evaluation of multiple initial structural options. The Visual BIM Query Language (VBQL), a block-based visual tool, specifically targets 5D-BIM activities related to building costs and time. Implemented via the Java-based BIMcraft

[2] https://docs.oasis-open.org/legalruleml/legalruleml-core-spec/v1.0/os/legalruleml-core-spec-v1.0-os.html

CC BY

prototype, it uses selection blocks to retrieve model information, with the BIMfit toolbox for queries and OpenBlocks for creating visual programming elements (Wülfing et al., 2014).

The Visual Code Checking Language (VCCL) is a node-based VPL designed for creating checking and verification routines, distinguishing it from query-focused languages (Preidel & Borrmann, 2015). While it includes querying, its primary goal is to embed building code knowledge for compliance. Despite its conceptualisation and implementation reflecting building code rules, a lack of practical application may indicate implementation challenges. The KBimCode Visual Language (KBVL), a flow-based VPL, aims to simplify the conversion of regulatory rules into machine-executable code (Kim et al., 2019). Built upon previous KBimCode work, it automatically generates KBimCode for execution. Additionally, Ghannad's work demonstrated combining VPLs (specifically Marionette within Vectorworks) with LegalRuleML to automate compliance checks for building designs, leveraging the existing engine and building model concepts of the proprietary authoring tool (Ghannad et al., 2019).

2.3 Compliance checking systems in practice

The increasing need for checking building information models against various criteria has led to the development of numerous platforms and applications over time. This software differs in focus, rule-definition approaches, and ability to create new checking rules. Certain applications are suitable for the preliminary design phases, helping achieve compliance of models before the submission of permits, whereas others are integrated into the municipalities' permit systems and can formally enable the pre-check as well as validate the model in the submission process.

Solibri Office (formerly Solibri Model Checker) is widely recognised for having extensible code-checking capabilities. Its rule checking system is based on hardcoded rule templates that can be adapted through parameters and is used for preliminary checks. Despite being used to check accessibility in an early Norwegian initiative (Eastman et al., 2009), the rule templates are still limited due to their hardcoded nature (Lee, 2011). CYPEURBAN has similarities with Solibri in terms of providing a hardcoded configurable set of rules; still, they are created based on permit regulations, which enables a more accurate verification at early stages for building designs[3]. It can also be used by municipalities to inspect the models and check within the permit process. The scalability of the software, however, is compromised by the lack of flexibility of the rules and its hardcoded nature. ACABIM also allows early-stage design checks, integrating New Zealand Building Code documents and supporting LegalRuleML to enable the creation of new regulations, though not all regulations are easily convertible to this format (Amor & Dimyadi, 2021; Eaton, 2021), which can compromise its scalability potential.

Tekto BIM engine, partially developed with the municipality of Salvador, Brazil, demonstrates how commercial software can incorporate local terminology for compliance checks in permit processes (Aida et al., 2022). However, its lack of new rule creation capabilities limits adaptability. Other software, like CorenetX[4] for Singapore and CheckerBIM for Geneva municipality, was specifically developed for permit processes but faces structural limitations for broader implementation[5]. These examples highlight the critical need for scalable, flexible, and adaptable compliance checking solutions that can accommodate diverse and evolving global building regulations.

A main limitation for the implementation of compliance checking software within the building permit processes resides in 2 factors. First, the closed and hardcoded approach for the implementation of rules implies less flexible and transparent rules. In this way, while the flexibility imposes a lack of scalability on the system, the lack of transparency can result in legal problems for the public administration. The second factor is the need to adapt to local permit terminology. Parameters such as the height of the building can be calculated differently in different municipalities, and these differences need to be implemented in the software, which can be unfeasible for software that is not prepared for this.

[3] https://info.cype.com/en/product/cypeurban/

[4] https://info.corenet.gov.sg/

[5] https://www.ge.ch/utiliser-bim-building-information-modeling/je-depose-ma-demande-autorisation-construire-bim

CC BY

3. METHODOLOGY

3.1 Overall considerations

The methodology of this work was fundamentally guided by Design Science Research principles, namely, to generate design knowledge aiming to solve real-world problems (Vom Brocke et al., 2020). Thus, the core workflow adopted in this work, based on (Peffers et al., 2007), considered the main steps of problem identification, definition of objectives (requirements), design and development and demonstration and evaluation. These steps were not only used for the overall work, but also for its components, being the practical workflow a series of embedded development cycles. As shown in Figure 1, the process was not followed as a complete linear process, since the choice for specific technical solutions or the result implied constraints that demanded the revision of requirements, generating a cyclical process. As an example, the need to retain the state of the blocks in the code editor demanded changes in the whole information flow, via Application Programming Interface (API) to database to incorporate the XML information that defines the blocks and their positions in editor.

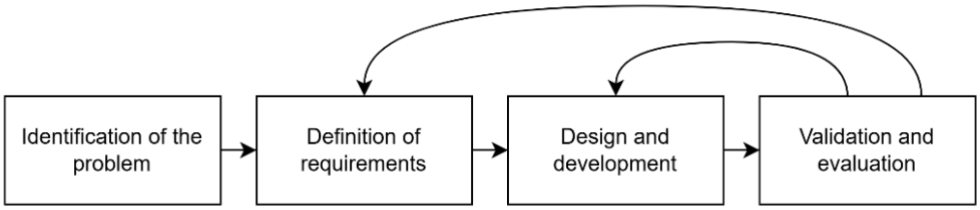

Figure 1: Main methodological workflow for the creation of the application.

The identification of the problem in this work was initially defined by the close contact and cooperation with municipalities and their technicians in efforts to promote the digitalization of building permits with BIM, exemplified by the works (Mateus et al., 2024; M. Santos et al., 2022; M. F. de S. Santos, 2021). These interactions occurred via work meetings, serving as input for empirical knowledge, and not consisting of structured input. However, they firstly demonstrated barriers to the adoption of BIM for building permits, namely low technical BIM knowledge, related to its tools, data models and programming, second early versions of the VPL were also presented, with feedback indicating potential of the approach. Other works already focused on a similar gap, mostly relating to a broader BIM adoption scenario, with the definition of VPLs to execute queries and data extraction or more generic compliance checking. Works such as (Kim et al., 2019) demonstrate an instance of a solution where VPLs are used for building permits, however it also highlights other limitations of the proposed solutions so far, since the solutions are not available for use also being of closed source, limiting their reuse and adaptation for other scenarios. Despite focusing on VPL as one of the components of the object of this work, the intention of the work is not to focus specifically on the benchmarking of this component, thus, it comes as a subproduct of the whole architecture, and a demonstration of how to bridge different levels of abstraction with incorporation of the permit's terminology. Consequently, a validation step was only considered as part of future work.

On a fundamental level, this work considered the conceptualization of rule checking as defined by (Eastman et al., 2009). Rule interpretation is object and subject in this work, and it is provided by a VPL and the incorporation of concepts into a domain model; the preparation of building model information is defined in a checking engine, that absorbs the model information via mapping that is defined in the domain model. Rule execution is defined also at the checking engine, that gathers all the rules defined by the VPL and runs them against the model information, and the report is generated as an output of the previous checking process. This work also adopted a stack of abstractions concept to promote a modularization from the lower level of the formal scripting programming language that executes the rule checking, and the data model from IFC, to the higher level of the VPL. This strategy intended to minimize the amount of hardcoding to be done when adapting or expanding the application. However, it is still considered that some amount of work needs to be done in coding, depending on the level of the layer that you need to change. For these reasons, the methodology of this work can be specified in terms of its main conceptual components: domain model; VPL and the architecture of the application. The considerations on scope and evaluation approach are in 3.5.

CC BY

3.2 Methods for domain model

The use of building information models to execute compliance checks within a specific domain has several challenges. One of the most relevant resides in the different terminologies adopted by the building model and the specific domain. While the building models adopt a general terminology that can be transversally incorporated in construction subdomains, it imposes an additional level of specialisation of its elements when they need to be mapped to more specialised concepts of these subdomains (Schranz et al., 2026). This can be clarified with the different definitions for building height presented in Table 1: while the height for the IFC is defined between the surface of the floor to the surface of the superior floor or roof, the Portuguese definitions use the threshold level as the lower reference and also disregard chimneys, for example, to define the top level.

Table 1: Different definitions for building height.

Building height as in IFC 4.0.2.1	Building height, as in Portuguese Decree law 05/2019
Standard gross height of this building, from the top surface of the construction floor to the top surface of the construction floor or roof above. Only provided if there is a constant height.	The height of the building is the vertical dimension measured from the threshold level to the highest point of the building, including the roof and other built volumes on it, but excluding chimneys and accessory and decorative elements, plus the elevation of the threshold, when applicable.

Works, as the one presented in (Lee, 2011), execute the mapping between these two terminologies by defining a domain model. It incorporates the specialised terms as constructs and serves as the mechanism to map these to the IFC data model, also considering some strategies, as the use of custom properties. This was the approach that was also adopted by this work, though it incorporated a specific methodology that is presented next, including mapping strategies that are also presented. The definition of a domain model naturally implies a definition of a domain, which in the practical case of this work was the one defined by representative documents from the Portuguese regulatory framework for building permits.

A representation of the steps followed to implement the domain model can be seen in Figure 2. The first step involved gathering representative regulations. Three Portuguese regulations were chosen: a) Regulamento Geral das Edificações Urbanas – RGEU, which is the main code to define urbanisation and building requirements at a national level in Portugal; b) Plano Diretor Municipal (PDM) of Vila Nova de Gaia, which is the main urbanisation document for the city of Vila Nova de Gaia; and c) Regulamento do Plano Diretor Municipal (RPDML) of Lisbon, being the main urbanisation document for the city of Lisbon. These documents were considered representative of the types of documents that define requirements to be used in the compliance check for the building permits, because the RGEU covers the national layer, shared by every municipality, and the two municipal plans cover the layer where terminology and requirements change from city to city. In addition, the other regulations, mostly appearing at the city scale, are expected to represent specialisations of the selected ones, as defined by the (Regime Jurídico Dos Instrumentos de Gestão Territorial - RJIGT | DR, 2015).

After the definition of the documents, the most representative clauses needed to be selected. This process was important due to the large scope of the end-to-end approach adopted by this study. The definition of clause, for this work, was considered as the minimal formal subdivision of an article containing the necessary information to execute a verification. This definition is, by essence, subjective, but this represents already one of the challenges of objectively chunking a document that was not constructed objectively. In practice, some articles can be self-contained, while in other cases, the title of the section can be representative of the zone where a specific requirement is applicable, as the same article can refer to multiple other articles, annexes or regulations.

From the three documents, nine clauses were selected as representative. The selection sought to cover different types of requirements, different ways of expressing them (text and tables) and different sources for the checked value (model properties, semantic classification and geometry), as motivated in Table 3 (section 4.1). All nine clauses were implemented as visual rules in the application, whereas three of them are presented in section 5 for the sake of brevity. The set is representative in terms of the types of requirements found in the documents, and not a statistical sample of the Portuguese regulatory framework: it combines national and municipal scopes, requirements expressed in text and in tables, and checked values taken from properties, semantic classification and geometry (Table 3).

CC BY

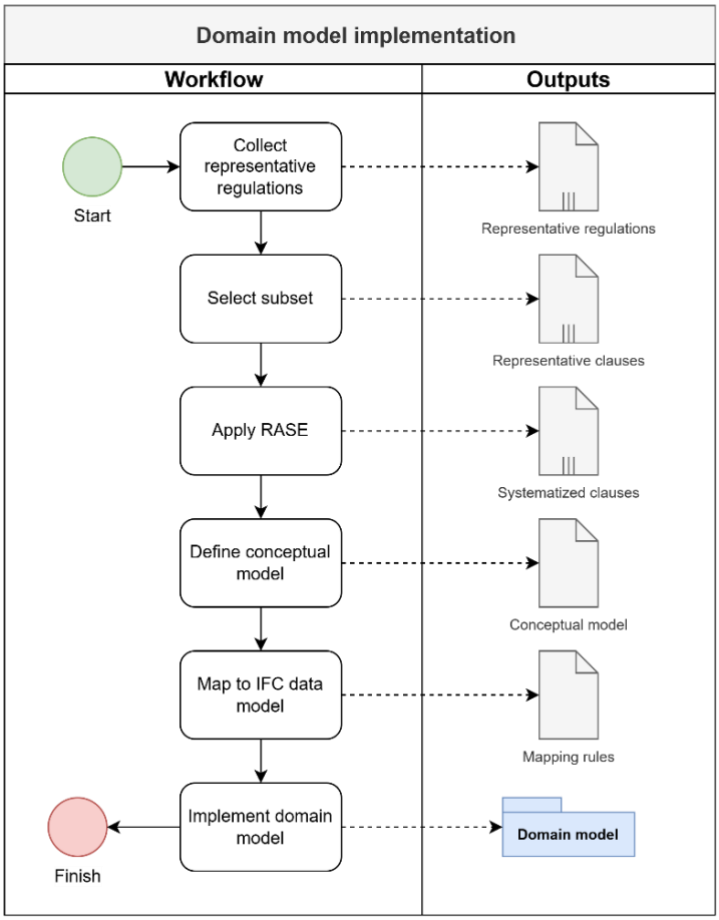

Figure 2: Process for domain model design.

To enable a systematic definition of the constructs and their relationships for the domain model, the RASE method (Hjelseth & Nisbet, 2011) was applied to every representative clause. This process helped define the elements, their properties and possible values objectively. However, to competently define the structure of the conceptual model, empirical knowledge was used, taking inspiration from the implicit relations demonstrated by the regulatory text and from data models such as the IFC. Following the definition of the conceptual model, the next step was to define the mapping between the model elements and their properties to the IFC data model. For the scope of this work, five different strategies were used to map elements from the IFC model to domain elements. These strategies are described in Table 2.

3.3 Methods for visual programming language

The main requirements for the definition of the VPL were the incorporation of the local concepts as defined by the domain model as well as the ease of use. VPLs of existing solutions often can be adapted at a certain level to create specialized functions. This strategy was explored by works such as (Ghannad et al., 2019) that uses Marionette, from the application Vectorworks, as a way to implement its conceptualization of the constructs of the VPL proposed, however this implies a vendor locking solution. Other open tools exist to implement VPLs, such as PyFlow, however, interactions with technicians from the municipalities at early stages of development suggested that the block-based alternatives of coding were more well accepted. Other collaborative projects such as n8n already use a VPL as the main interface, however, a severe adaptation on the platform would be needed to adapt it for the purpose of this work, since its architecture and components were conceptualized for very distinct use cases. For this reason, Blockly was chosen as the main tool to develop the VPL since it is a library focusing on block-based VPLs and that has a stable development, being used for well-known projects such as Scratch (Seraj et al., 2019).

CC BY

Developing a VPL using Blockly demands the definition of the vocabulary, meaning the set of available blocks, and how the generated code will run (Pasternak et al., 2017). Also, visual styles can be used when creating the visual aspects of the vocabulary, implying different code generation approaches. In the case of block languages, the visual styles most frequently used are: icons, that rely on images to represent the function of the block; natural language, that presents standard written sentences to represent the function of the block; computer language, that translates the vocabulary of a formal language into the blocks (Pasternak et al., 2017). For this work the vocabulary definition considered the practice of established block-based languages, such Scratch, that uses formal language structures, with an icon-based approach, to represent the entities of the domain model.

Table 2: Strategies used to map the domain model to IFC.

Use of native IFC properties and property sets	Mapping with one-to-one correspondence between the elements or properties from both models. In the example of this work, this strategy was used to map the building entity of the conceptual model to the IfcBuilding entity from the IFC data model. Yet, this strategy is very limited, since sometimes properties with the same name can have different definitions, as exemplified in Table 1.
Customised properties and property sets	Used when a correspondence can be established with the creation of a custom property or a custom property set. Despite the ease to require properties as custom ones, this approach has two main caveats: i) it can generate a long list of properties that must be included in the model, turning the process of modelling daunting; ii) properties based on physical measurable aspects of the building can be incoherent with the actual geometrical representations of the building, causing inconsistencies. This last problem is also a reality for native quantity sets, but since the quantities are usually calculated from the geometry while exporting the IFC, this problem is minimised.
Classification systems	Codes from a classification system are used to map elements from the domain model to the IFC data model. It can be particularly useful to map, for example, IfcSpaces to different types of rooms, such as a kitchen and bedroom. For this work, the Portuguese system SECClasS was used. This system is a Portuguese implementation of the Uniclass systems, being for the most part a translation into Portuguese of the terms. However, this strategy demonstrated that even types of rooms, that are mentioned in the regulations, are not present in SECClasS. For this reason, custom codes were created to explore this alternative.
Geometric processing	This strategy aims to obtain property values of the domain model by processing the geometrical representation of elements in the IFC model. Ultimately, it can guarantee the coherence of the information attributed to the domain model with the geometry of the IFC and minimise the number of customised properties, since it can determine information by implicit geometric aspects. However, it can bring challenges to be implemented, due to the number of geometric representations possible in the IFC, which imply technical challenges to create consistent algorithms to process.
Attributes	In this strategy, attributes from the model are mapped to attributes of IFC elements. Differently from the mapping based on properties, there is no possibility to create custom attributes, therefore, they must be used with coherence with its IFC definition.

Code generation enables the conversion of blocks into a string of code that can be executed. In Blockly, each block is associated with its own formal code which is done by JavaScript-based classes denominated block-code generators. By default, the library provides built-in generators for Python, among other languages, but these generators only cover the main aspects of the Python language. To create a DSL capable of generating Python code focused on executing compliance checks for building permits, block-code generators were developed to create specific Python code that integrates classes and methods from the compliance check library also developed within this work.

3.4 Methods for architecture of application

For the architecture definition, the main steps of the process defined in section 3.1 were the requirements and design and development. The first was framed by the need to use open-source technology as well as the modularity for the integration of the main components. The requirements were also impacted by the evaluation of the available frameworks and stacks to determine the most suitable to achieve these goals. Other aspects were also assessed, such as the popularity of the framework, which impacts the amount of available content and support. Then the

CC BY

design and development involved the integration of technologies, definition of data structures and interoperability mechanisms, such as APIs and serializers.

3.5 Considerations on evaluation approach and scope

Following the steps of Peffers et al. (2007), this paper reports the design and development cycle that ends with the demonstration of the application. The evaluation carried out is technical, focusing on the correctness of the execution of the rules and on the flexibility of the visual language for a set of representative clauses, selected to cover different regulatory scopes (municipal and national), conditional structures and ways of obtaining the checked parameter (section 5.1), and not including evaluation with end users at this stage. The execution correctness is assessed by comparing the values derived by the checking engine with the values obtained through independent measurement in a separate model viewer. The flexibility of the visual language is considered in terms of whether each clause could be expressed with the existing blocks and domain model concepts. The comparison was carried out on a model based on an actual architectural design planned for Vila Nova de Gaia, with 10 floors and 20 dwellings, prepared according to the information requirements of the application (section 5.2). The results are presented in section 5.3 and discussed in section 6.

4. APPLICATION DESIGN

This section presents the design and its results as the developed application, focusing first on the main components as presented in section 3, a further exploring other application components in the architecture section.

4.1 Domain model

From the document analysis of the three documents from the regulatory framework, nine clauses were considered as representative; they involve different types of requirements and different manners of presenting the requirements. This supports the expectation that the constructs deriving from these requirements are referred in other clauses and documents as well, such as the example of the number of building storeys, that was one parameter presented in one of the representative clauses for Vila Nova de Gaia but is also used other 23 times to define requirements in the same document. Taken together, the nine clauses selected span the national scope (five clauses from the RGEU) and the municipal scope (two clauses from the PDM of Vila Nova de Gaia and two from the RPDML of Lisbon), requirements written as text and as tables, and values obtained from model properties, from the semantic characterisation of spaces and from geometry. All the clauses and their motivation to be considered as representative are presented in Table 3.

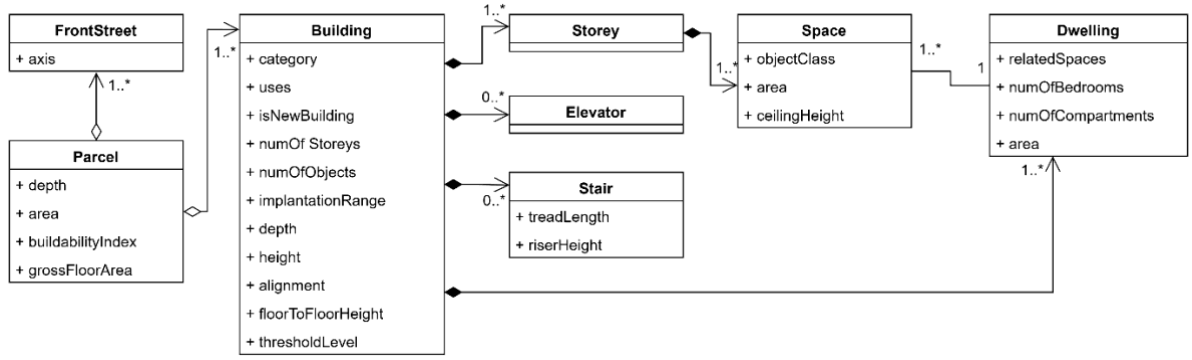

Figure 3: Conceptual model created based on representative clauses.

4.1.1 Applying RASE and conceptual model definition

Through the application of the RASE process, the main elements of the representative clauses were obtained. They were then processed using empirical knowledge to define a conceptual model, presented in Figure 3. This model represents a partial model of the Portuguese permit concepts, correspondent to the clauses analysed, and is composed of 8 entities and 25 properties. The use of empirical knowledge was needed due to unreadiness of the constructs as they are presented in the natural language and therefore as they are captured by RASE as a markup

methodology. This also can be exemplified by the property "riserHeight" of the element Stair in the conceptual model, that roughly appeared in natural language as "The risers of collective residential buildings must have … maximum height of 0.193 m". Thus, a direct conversion of what is captured from natural language needs to pre-process into a logical and modular structure, which also the process that is demanded on the creation of ontologies.

Table 3: Representative clauses and the motivation for their selection.

Reference	Text	Motivation
RGEU - ART. 46, § 7	The steps of staircases in multi-family residential buildings shall have a minimum tread depth (going) of 0.25 m and a maximum riser height of 0.193 m. However, in buildings of three, four or five storeys, and wherever no lift is installed, the minimum tread depth (going) shall be 0.280 m, and the maximum riser height shall be 0.175 m. The dimensions adopted shall remain constant throughout the flights between consecutive floors.	Clause that defines requirements for a component of the building, in this case, stairs, but has applicability conditions on physical and typification aspects of the construction.
RGEU - ART. 65S, § 1	The minimum floor-to-floor height in residential buildings is 2.70 m (27M), and the minimum clear ceiling height may not be less than 2.40 m (24M).	Determines an internal constraint (height floor-to-floor) that represents a national-level height requirement.
RGEU - ART. 65, § 2	Exceptionally, in entrance halls, corridors, sanitary facilities, pantries and storage rooms, the ceiling height may be reduced to a minimum of 2.20 m (22M).	Presents a parameter (ceiling height) constrained by a semantic characterisation of spaces.
RGEU - ART. 66, § 1	The number and area of habitable rooms in dwellings may not be less than those indicated in the following table. (Table)	It's a clause that defines requirements within tables.
RGEU - ART. 67, § 1	The gross floor areas of dwelling units shall have the following minimum values: (Table)	Present national concept of dwelling and its appropriate dimensions.
PDM OF VILA NOVA DE GAIA ART. 24, § 1, A)	Facilities directly associated with agricultural; livestock or forestry operations must comply with the following building conditions: a) Buildings must have a maximum height of 2 storeys and a maximum height of 6.5 m, except for duly justified technical installations;	Present height and number of floor constraints.
PDM OF VILA NOVA DE GAIA ART. 42, § 1	New main buildings shall be sited within a 35 m strip adjoining the public space, without prejudice to the provisions of the following paragraphs.	Define implantation depth constraints.
RPDML ART. 42, § 7, E)	In urban layouts of type D – detached houses, construction, alteration and extension works shall comply with the following rules: i) 1.0 on a plot or parcel with an area of less than 150 m²; ii) 0.7 on a plot or parcel with an area equal to or greater than 150 m², a minimum of 150 m² of floor area always being permitted.	Define constraints for a building index.
RPDML ART. 43, § 1	Without prejudice to the provisions of the following paragraphs, the maximum building depth (measured along the party walls), excluding balconies and cantilevered projections, is 15 metres, except for hotel establishments and public-use facilities, where it may reach 18 metres.	Define constraints for the building depth.

4.1.2 Mapping strategies, information requirements and geometric data retrieval

Mapping was executed considering the strategies as defined in section 3.2, and Figure 4 presents examples of how they were applied to different elements of the conceptual model. Depending on the concept, the value is obtained directly from model properties, from the semantic characterisation of spaces or through geometric processing, and this choice determines what information the IFC model must provide. On the geometric processing strategies, for the scope of this work, not only the building heigh was implemented, but also the building depth and implantation range, since they can be represented by the biggest linear distance between elements in a certain axis or alignment. The retrieval of these elements were based on the semantics of IFC, which also blended mapping strategies, since for the calculation of height, a threshold level must be defined, and this implied in a mapping using the attribute strategy, however the other calculations were done using the geometrical representation of the objects obtained with the methods following described.

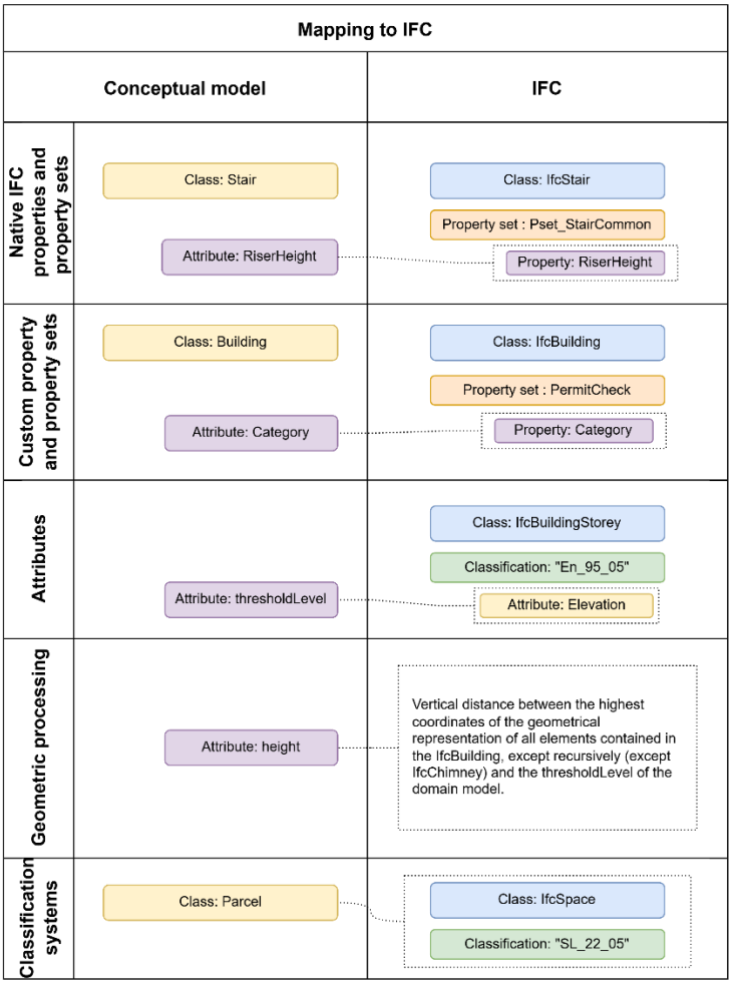

Figure 4: Examples of mapping between conceptual model and IFC.

The information requirements therefore derive directly from these mapping strategies and are summarised in the Level of Information Need tables presented in Annex I. An IFC model needs to follow these requirements to have its information well incorporated into the domain model and therefore be used to execute the compliance checks. To retrieve this information, together with the geometry needed for derived values, a library called "CHECKIFC" was developed to simplify the process of retrieving IFC information via query along with geometric information using a simple geometric engine. CHECKIFC integrates the IfcOpenShell parsing, mesh generation and selection capabilities with the mesh processing capabilities of Trimesh. The main class of the library is denominated CheckModel, and it comprises methods to load and select elements from the IFC model. The loading workflow is presented in Figure 5, and it starts with the parsing of the IFC using IfcOpenShell. Then, it creates Trimesh objects

CC BY

with every shape representation of IfcProducts and stores them in a dictionary included in a CheckModel object. Finally, the trimesh objects corresponding to each IFC element are gathered from the mesh dictionary. The last step is the creation of a new CheckElement, which is the object responsible for putting together the Trimesh and IfcOpenShell APIs.

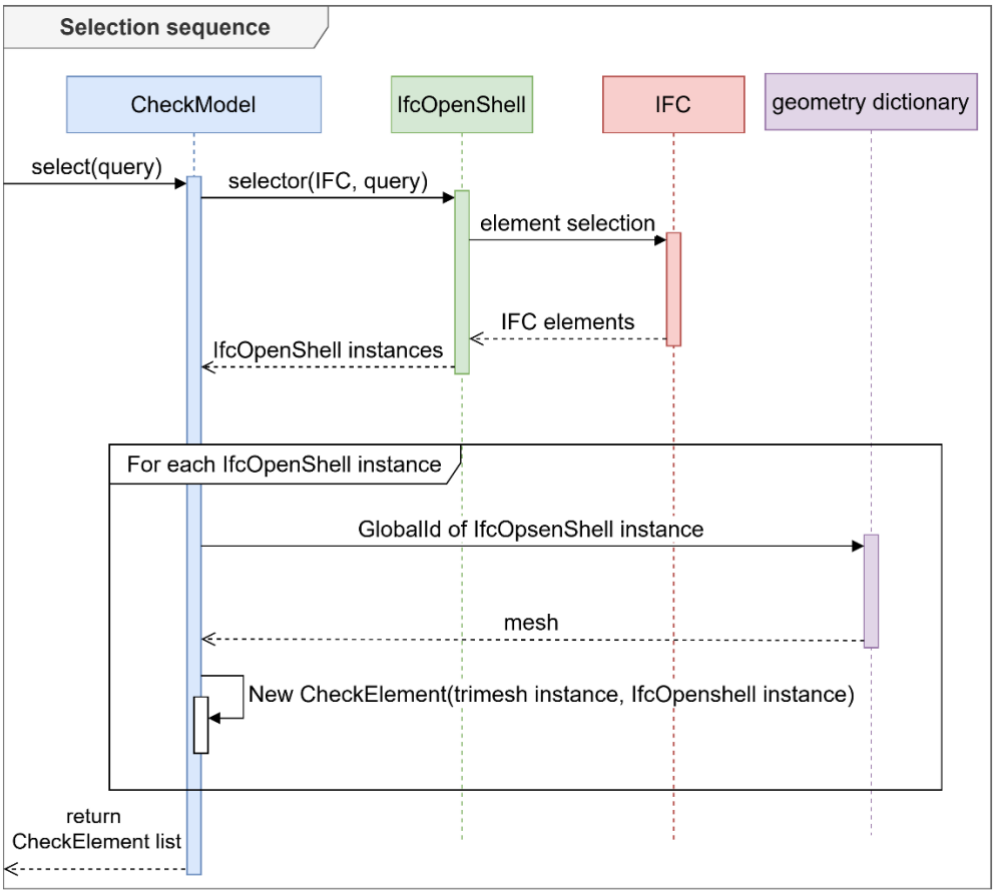

Figure 5: Selection sequence implemented in the CHECKIFC module.

Mapping definition, alongside with the system to retrieve information, enabled the implementation of the domain model as a series of Python classes. The classes of the model, in the majority, represent the classes of the conceptual model as it was originally conceived. However, additional classes were created to allow a better modularisation in the implementation. For example, additional methods such as *highest_point()* were created to use within the calculation of building height but can also be used in other additional parameters if needed. Properties were also implemented as methods rather than attributes, as in the example of the area properties, since their value comes from geometric processing.

4.2 Visual programming language

As defined in Section 3, the core aspects of VPL development relied on the vocabulary and block-code generators. This was complemented with components at the level of the user interface that managed how the information was input and processed by the user. One aspect of the design that was defined posteriorly was adoption of atomic sentences, as defined in Park et al. (2016), while implementing a design based on computer language, icons and natural language styles (Pasternak et al., 2017). These choices allowed us to define a simple yet flexible language capable of creating rules that are more complex than the parameter based or with the use of Component-requirement logic. Local building permits were also central to this approach, enabling domain experts to code using their own local terminology.

CC BY

In a technical aspect, React was used as the main enabler of the UI and the aggregator of the remaining frontend components. React is a JavaScript-based library that facilitates the development of dynamic user interfaces. Similarly to Django, it also adopts a modular approach, which in turn is defined by named components. Additionally, the core implementations related to the visual language were done also with Blockly, that enables the development of the necessary elements to code using the VPL, such as the editor.

4.2.1 VPL design and vocabulary

Since the abstractions of the object-oriented languages can be hard to understand by domain experts at the municipalities, the icon style was used on blocks to represent objects related to the building. The natural language style was used to represent properties of the objects in the model. This choice was made to facilitate the understanding of the relations between blocks. Another important aspect considered when creating the vocabulary was the need to represent complex rule structures. Then the computer language style was also used to implement flow control entities, exemplified by the if statement. In summary, all of these styles were considered to define the final vocabulary of the visual language.

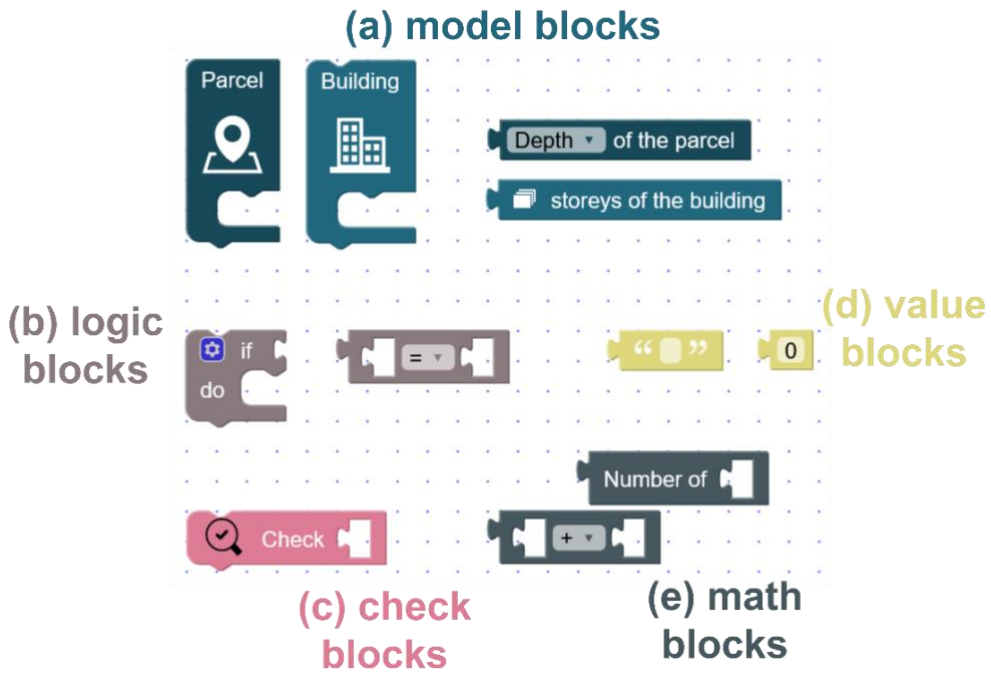

Figure 6: Types of blocks.

Five categories of blocks were developed, and are presented in Figure 6:

a) Model blocks: represent the entities and properties from the domain model. The main blocks within this category are the entity blocks, that are responsible for looping through all the elements of their designated type. For example, and as demonstrated in Figure 6, the "Building" block is incorporated within the "Parcel" block, representing that all the buildings within all the Parcels are going to be looped through.
b) Logic blocks: logical related operations and conditions, brought from the formal languages and are capable of giving flexibility to the codes generated. The block for the if-statement can filter the block loops, limiting, for example, the loop through storeys that have a specific gross area. The other blocks of this category bring conditional operators, that can enable the filtering together with the if-statements as well as be integrated in the checks.
c) Check blocks: the blocks that represent the main verifications to be executed. Two blocks are implemented in this category. The "Check" block represents a mandatory condition that must be followed and has a positive or negative result. The "Alert" block represents a condition that must be followed but can be the object of a subjective analysis if not passed. This last block was created specifically to deal with this type of situation that is frequent in the Portuguese regulatory framework.

CC BY

d) Value blocks: represent values, that can be numerical, textual or even categories that can be integrated together with the other blocks to filter the elements, compose the checks or create the alert messages.
e) Math blocks: represent the arithmetic operations and counting functions that can be integrated with the other elements to give more flexibility to the rules.

4.2.2 Block-code generators

The framework proposes uses the Blockly features to generate formal Python code under the blanks while using the VPL, relying on code generators that are integrated within the library. These code generators allow the conversion of block components into Python strings, considering also the block connections, that can serve as input to other blocks. This allowed the generators to be configured to incorporate the elements of the domain model, incorporating all its classes, attributes and methods. Figure 7 demonstrates one example of code generated by a group of blocks.

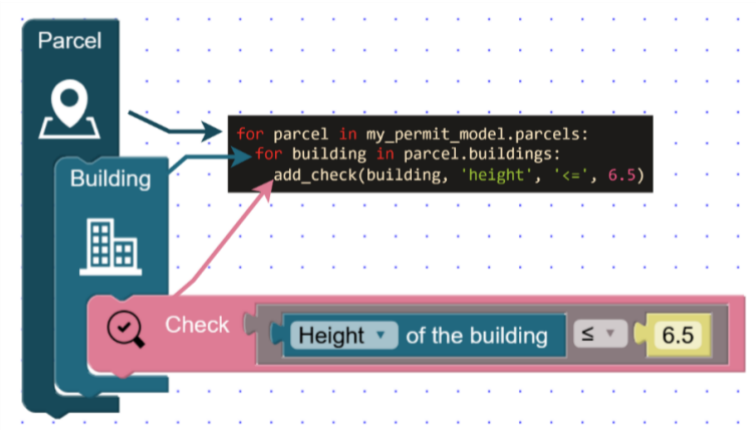

Figure 7: Code generation.

4.2.3 Code editor and IFC model viewer

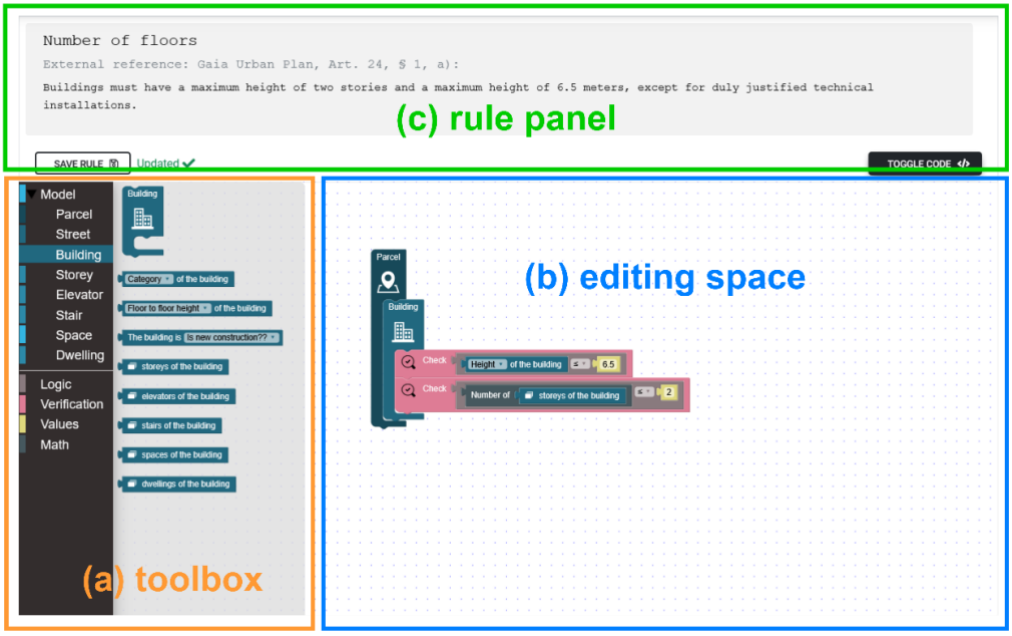

Figure 8: Main parts of the code editor.

To enable the use of the visual language, an editor was implemented, which was considerably facilitated by the Blockly library. The main elements to be defined in the editor were its toolbox and visual aspects. The final version of the editor can be seen in Figure 8, and features the following elements: (a) available blocks and their categories; (b) text of the verification; (c) button to save rules; (d) main editing space; (e) button to toggle the code panel, where the Python code generated by the blocks can be inspected. Within the editor, all the existing blocks can be associated in different ways to create multiple different combinations. The categories of blocks, presented in 4.2.1, as well as a consistent colour palette, were implemented to provide the visual intuitiveness necessary from VPLs.

4.3 System architecture

4.3.1 Overall architecture

The overall application architecture, presented in Figure 9, was designed to give the proper foundations to achieve all the application requirements. Its main structure is conventional for web development, and is composed of two main tiers: frontend, which is related to the components that are executed at a browser level, and backend, that has components that are executed at a server level. LiMA Blocks contains the main components for the creation of the machine-readable regulations and the model viewer on the frontend; on the other hand, the compliance checking engine and database are located on the backend. This choice was made considering the technical constraints of the chosen libraries, as well as to use the full capabilities of the server to execute asynchronous process-demanding activities, such as the IFC model geometry processing. Finally, an API connects these two tiers by enabling information exchange about the code created with the visual language, IFC models, and reports. All the components described in this section are available in the public repository of the project (https://github.com/Muniz1994/LimaBlocks), which allows the implementation to be inspected, reproduced and adapted.

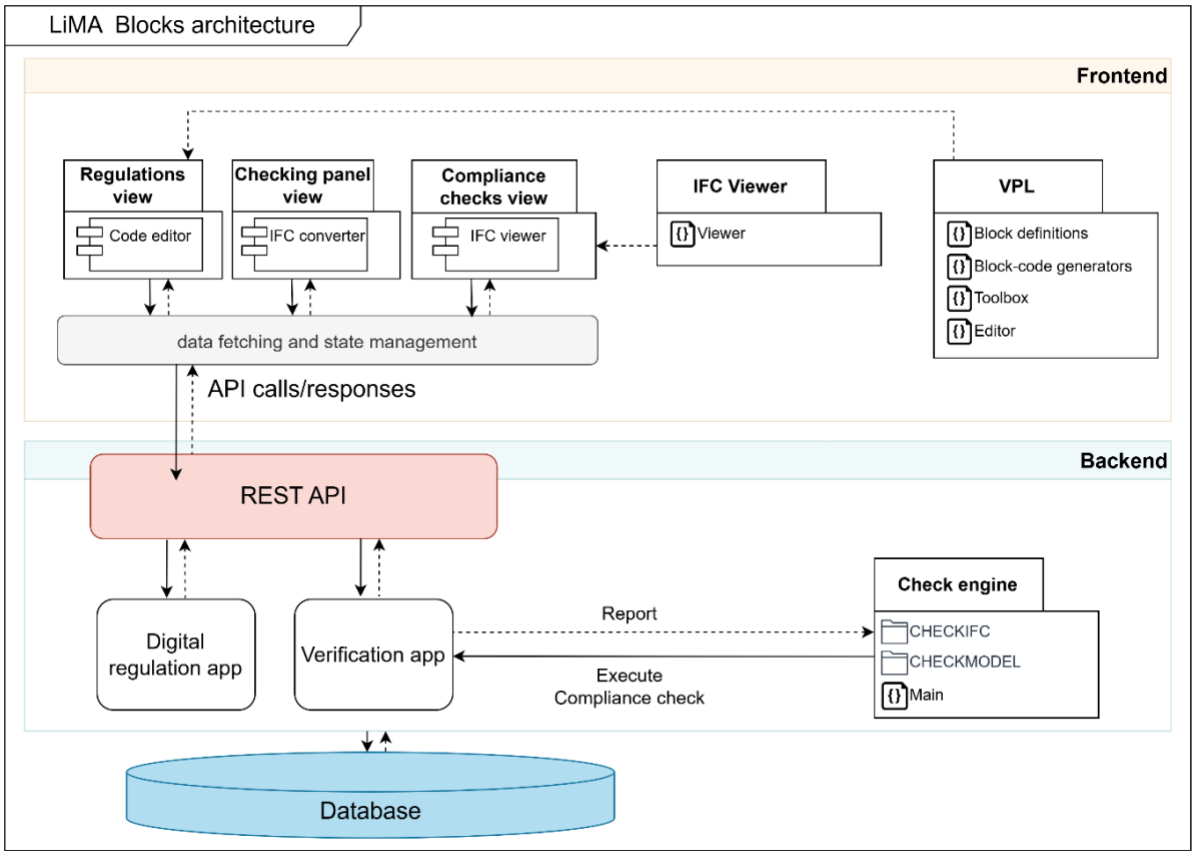

Figure 9: LiMA Blocks architecture.

Technological choices, such as the use of React and Django libraries, were made due to the technical scalability and ease of development that they bring to the solution, since both are based on modularised approaches. Yet, these enable powerful development capabilities, as well as security features that are incorporated into the final web application. Due to its architecture, activities involved in the application's design and development were comprised in two main design and development blocks: (i) backend and compliance check engine; (ii) frontend and VPL.

4.3.2 API and information exchange

Considering all the components previously demonstrated, a simplified overall sequence for the workflows of the application is demonstrated in Figure 10. Mainly, the operations occur via interaction of the user with the application via its interface and between the two main tiers of the application as stated in 4.3.1. The latter is done mostly via an API, that need to convey all the information in a structure and bidirectional way. REST (Representational State Transfer) is an architectural style for APIs that includes properties such as usability, simplicity, scalability and extensibility (Li & Chou, 2011). It can exchange information in different formats, JSON (JavaScript Object Notation) being one of the most popular, since it's readable by humans and machines, as well as being programming language-agnostic. The format is also used to serialize information on city models, such as CityJSON (Ledoux et al., 2019), and is currently being used as the serialization format for the IFC5 development (van Berlo et al., 2021).

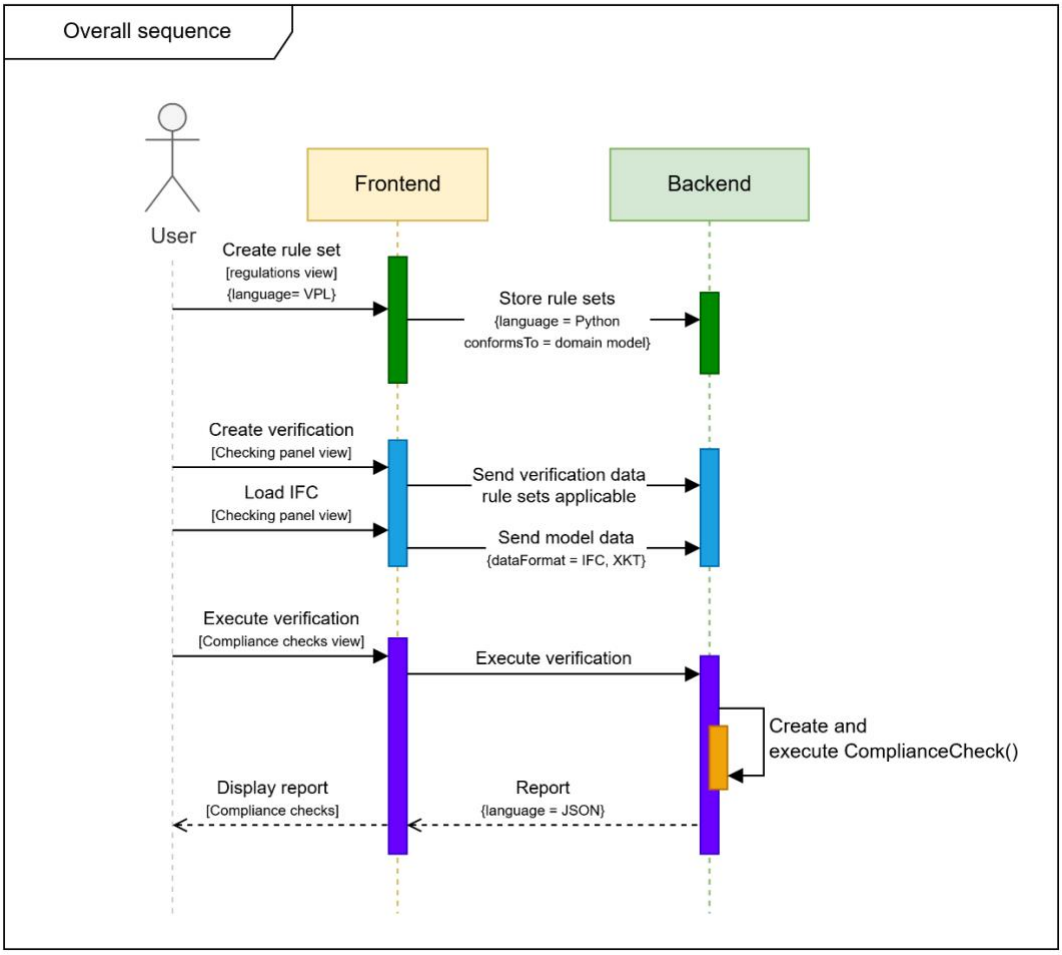

Figure 10:Overall sequence of operations to create rules and execute compliance checks.

For these reasons, a REST API was created as the main information exchange mechanism for the application, transmitting all the rule, report and model information between the Django-based backend and the React-based frontend. This choice also directed the definition of the machine-readable formats for the regulations and the reports, that are also serialized in JSON. The API is composed of four main endpoints, by means of contact between

the frontend components and the backend components and are defined by a URL. Rules, regulation and verification endpoints have the main function to exchange information with the backend database management system, creating, updating and retrieving information. The execute verification endpoint, however, has the goal of executing the check engine in the backend and receiving the report.

For the purposes of this work, a regulation is defined as a set of machine-readable rules and their associated metadata that establish a link to existing legal regulations. The regulations are created within the frontend and are exchanged with the backend, to be stored or edited via API. With this in consideration, the main information, its structure and format needed to be defined. Mainly, the structure of the digital regulation was conceived to capture important parameters needed to enable the technical demands from the compliance check engine, presented in section 4. All this information from rules and regulations was serialized in JSON format, to be easily exchanged through the API. The information included in the regulations is:

a) Id: unique identifier of regulation;
b) Name: natural language name to represent the regulation;
c) Rules: comprise the list of rules;
d) Scope: indicative of the zone where this regulation is valid.
e) The rules, within the regulation, also needed to be defined. Their main components are the following:
    a. Id: unique identifier of rule;
    b. Name: natural language identifier of rule;
    c. External reference: natural language parameter to identify references in building code or regulations that this rule represents;
    d. Code: parameter that contains the logical rule in a Python textual format. This parameter is filled with code generated by the blocks of the visual language.
    e. Blocks: parameter that is filled with the XML representation of the blocks used to generate the code. It is ultimately used to retrieve the VPL code in the frontend, showing the block code and allowing it to be edited.

To allow the visualisation of the report in the frontend, a simple report format was conceived. It is composed of a list of JSON-serialised results for the rule check, each one containing the following parameters:

a) Id: unique identifier of the report;
b) Reference: The reference of the rule is verified.
c) Checks: a list of the atomic checks verified with their results and associated objects.

With this, the results can be displayed in a simplified manner to the viewer.

The remaining aspect of information exchange is related to components that are deeply described in the following section.

4.3.3 Backend and compliance check engine

Django is a Python-based web framework that contains high-level features to facilitate the development of scalable applications. For this reason, it was chosen as the main solution to implement the backend tier. By being based on Python, this framework facilitates the integration with libraries such as IfcOpenShell, therefore streamlining the integration of IFC parsing and processing capabilities. Another benefit of its use resides in the ease of implementation and management of databases and REST APIs, which are fundamental pieces of the architecture of the main application.

A Django project is structured in modular minor applications, which are packages that provide a specific piece of functionality. The applications, in turn, incorporate models that define their data structure and enable the management of this data with the database. For LiMA Blocks, two applications were implemented, as depicted in Figure 11: the digital regulation, which manages the data related to the development of the machine-readable regulations and their incorporated rules and the verification app, which is responsible for managing the verification information, as well as calling the compliance check engine. The implementation of the database engine was done through SQLite, which is the default solution for Django applications, though it makes the database easily replaceable. The database structure is automatically defined by the models and simply reflects the data structure of the digital regulation format and the report format.

CC BY

To enable an end-to-end development of the application, the implementation of compliance check capabilities was mandatory. Technical and design choices were made to guarantee the achievement of the proposed features as well as the integration with the other components. For that, the check engine was developed as a Python module inside the Django project, which can also be seen in Figure 9.

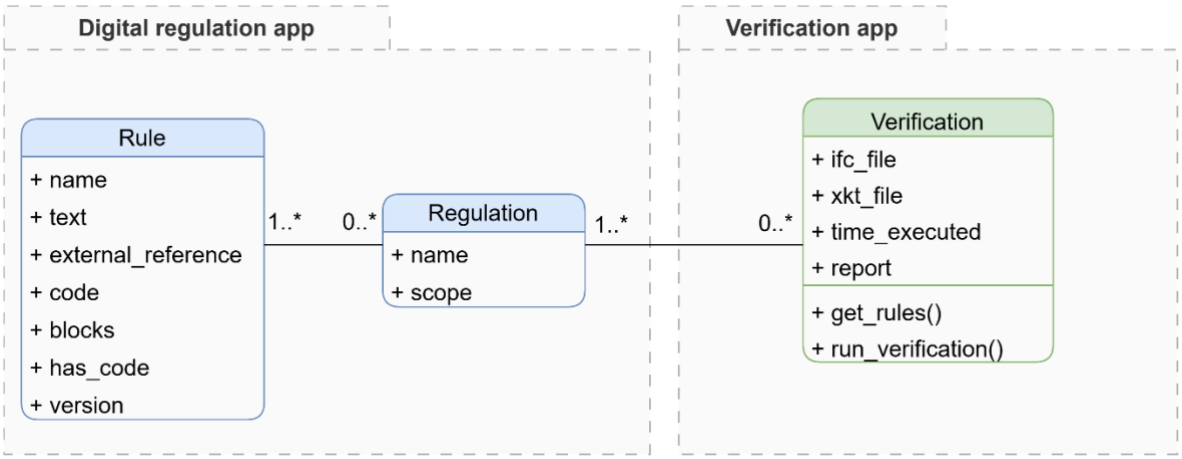

Figure 11: Django apps and database structure.

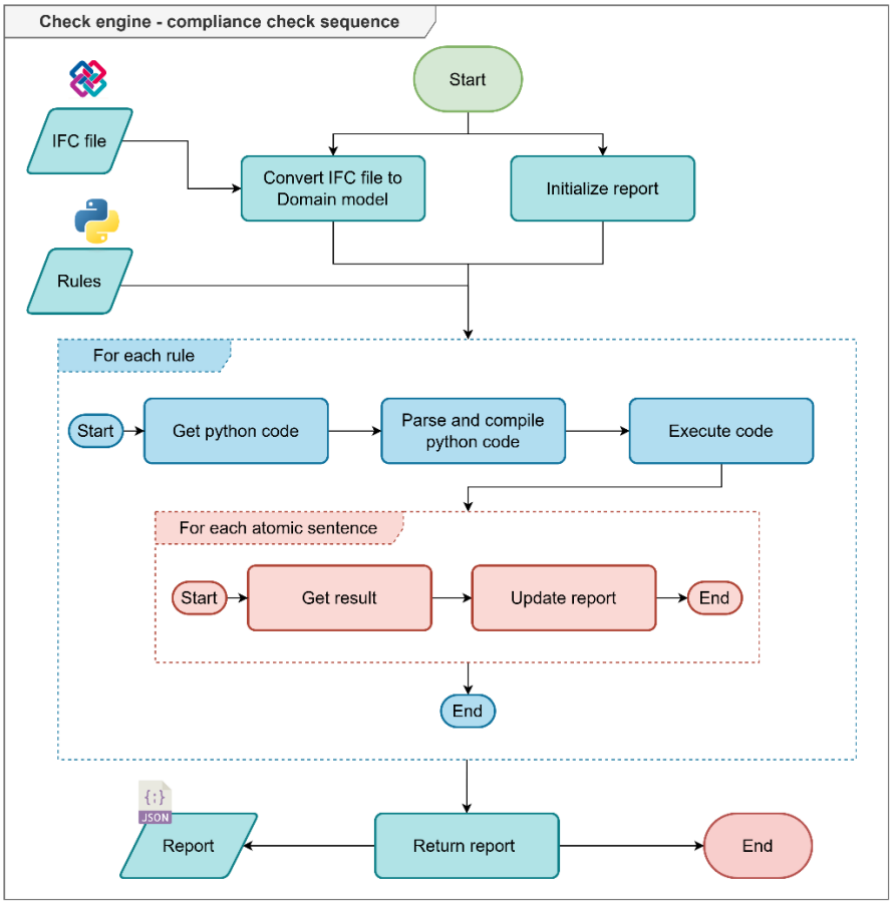

Figure 12: Compliance check workflow.

The overall checking workflow implemented by the engine can be seen in Figure 12. It starts with the conversion of the IFC model to the domain model, demonstrated in section 4.1, which maps the local building permit concepts to the IFC data model. Another initial step is the initialisation of the report that is going to be filled with the results of every rule executed. The next step is the interpretation, compilation and execution of the Python code, in textual format, associated with the rules. This step uses the Abstract Syntax Tree module to enable the parsing and interpretation of the textual code; at the same time, this code is executed in a different environment. For this reason, the converted IFC model, as well as the report, are passed as resources for the Python code, which is a practice that incorporates security measures to prevent the possibilities of code injection. Finally, every atomic sentence is captured to form the final report of the compliance check.

The concept of atomic sentences (Park et al., 2016), understood here as a declarative sentence that returns a Boolean condition, was used to capture the results of execution for each code associated with a rule, also associating it with the report. This choice is one of the few that also represent potential constraints for the design of the visual language. One of the main benefits of using the atomic sentences is to standardise the capture of the main elements needed to be present in the report. For this reason, a method named "add_check()" was developed within the check engine, and it assesses the atomic sentence and adds the main verified elements to the report. The method has the following declaration parameters:

add_check(subject, attribute_or_method, comparison, object)

- Subject: main element of verification;
- Attribute or method defines a characteristic of the element that can be represented through a simple attribute or, for more complex calculations, present in a method;
- Comparison: a logical operator or relation between the elements that can return a Boolean value;
- Object: the element that determines the value to be compared to. It can be a simple numerical or textual value, as it can also be more complex elements, such as other embedded logical sentences.

5. USE CASE

To demonstrate the capability of LiMA Blocks to execute all the stages required for a compliance checking system, as presented in section 2, a use case was developed. Therefore, the use case demonstrates the: a) use of the application to create rules using the visual language within the code editor implemented; b) preparation of the BIM model according to the information requirements; and c) compliance check and report within the application. Besides that, a manual verification of the rules was executed using an alternative model viewer. This enabled not only the comparison of results obtained with LiMA Blocks but also served as a basis for discussing the contributions, challenges, limitations, and potential of this work. For the sake of facilitating the comprehension, all the UI and the clauses presented within this work were translated into English. However, a Portuguese version of the UI is also implemented, since the use of local terminology can be compromised by the translation of terms and definitions to other languages.

5.1 Rule creation

The demonstration of the rule creation capabilities was done by implementing the representative rules within the application. Three of the nine implemented clauses were selected to represent three different dimensions: the first dimension is the regulatory scope, covering municipal plans, for Vila Nova de Gaia and Lisbon, and national regulation, for the RGEU, which differ both in terminology and in the way the requirements are expressed; the second dimension is the conditional structure of the rule, ranging from an unconditional threshold, to a branching condition evaluated on a semantic property of the element, to a tabular requirement resolved through a chain of alternative conditions: the third dimension is the way the checked parameter is obtained, ranging from a property directly available in the model, to a geometry-derived value that requires semantic exclusions, to a geometry-derived value obtained by slicing a volumetric definition. Each clause therefore exercises a different combination of these dimensions, and together they illustrate a sample of the rule forms that the language is intended to support, without claiming to exhaust them.

CC BY

First, a city-level clause from Vila Nova de Gaia's urban plan was chosen:

Vila Nova de Gaia - Urban Plan, Art. 24, § 1, a): "Buildings must have a maximum height of two stories and a maximum height of 6.5 meters, except for duly justified technical installations."

The implementation of this clause, presented in Figure 13, involves the use of "Parcel" and "Building" blocks from the model category, which means that all the buildings within all the blocks. Then, one check block is implemented to verify the maximum number of stories and another one to verify the maximum building height. Within the check blocks, blocks representing the properties of the building and blocks to determine values and comparisons are used. This case already demonstrates the simplicity of the block-based rule, which is attained by the design principles of the language and the use of permit terminology.

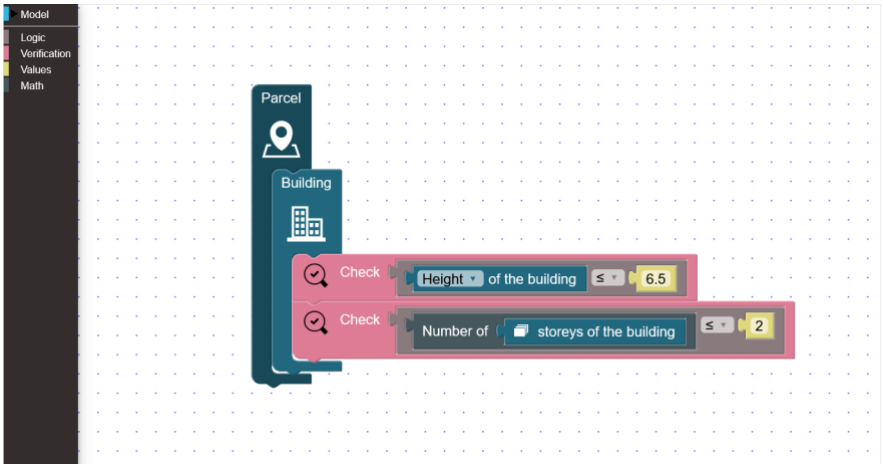

Figure 13: Implementation of clause from Vila Nova de Gaia.

The second clause is a city-level clause from Lisbon:

Lisbon – Regulation of Urban Plan, Art. 43, § 1: "Without prejudice to the provisions of the following paragraphs, the maximum depth of buildings, excluding balconies and cantilevered structures, is 15 meters, with the exception of hotels and public facilities, whose depth may reach 18 meters."

The visual code for this clause can be seen in Figure 14, and it has the particularity of using an if-statement to conditionally determine the checks based on the use property of the building element. Besides that, as determined in the text of the clause, the calculation of the depth of the building disregards "balconies and cantilevered structures", and this was already included within the depth calculations for the domain model, thus removing the need to determine this using the language. Some blocks also better describe the information about the parameter when hovered, as also demonstrated in Figure 14.

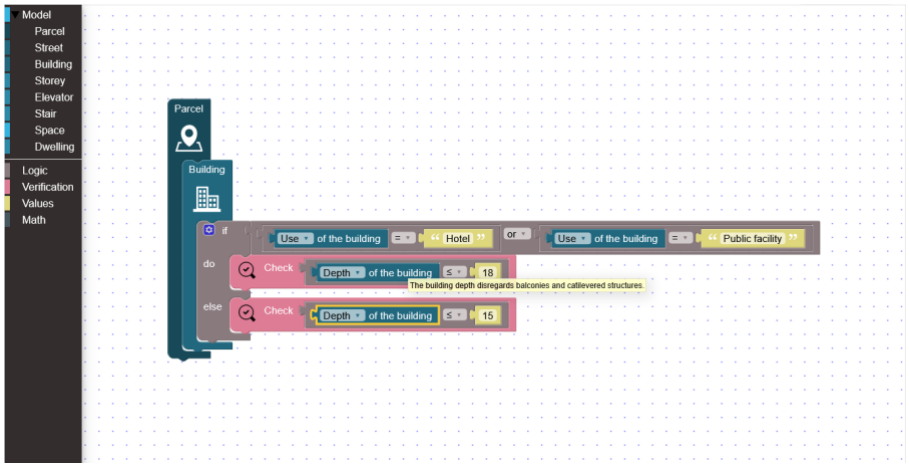

Figure 14: Implementation of clause from Lisbon.

The last clause chosen was one at the national level, being applicable for Lisbon and Vila Nova de Gaia:

RGEU - Art. 67, § 1: “The gross areas of dwellings shall have the following minimum values: {}”

Table 4: Table of areas for dwellings. Adapted from "Regulamento Geral das edificações Urbanas".

Type of dwelling	*T0*	*T1*	*T2*	*T3*	*T4*	*T5*	*T6*	*Tx > 6*
Gross area in square meters	*35*	*52*	*72*	*91*	*105*	*122*	*134*	*1,6 x Ah*

Unlike the previous clauses, this one has the requirements defined within a table. Hence, the implementation, as presented in Figure 15, contains a long series of conditions that are created using the if-statement block. This block can be expanded to embrace alternative conditions, “else-if”, that enable different checks for different applicable values.

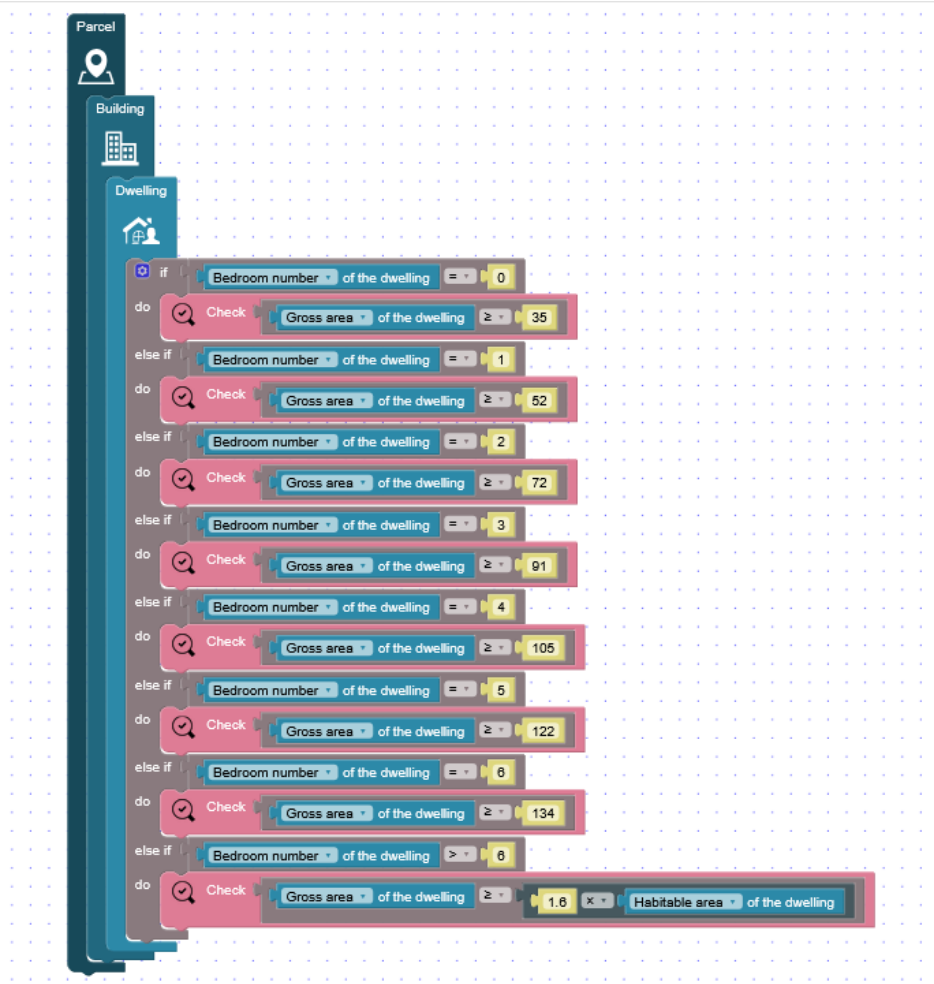

Figure 15: Implementation of clause from Portuguese national regulation.

5.2 Model preparation

For the demonstration of the code checking capabilities, a model was elaborated based on an actual architectural design planned for Vila Nova de Gaia. It consists of a 10-floor building, 20 dwellings and areas for services and commercial use. The internal architectonic aspects of the building were loosely conceived, not being compliant with most of the rules. The model was developed in a proprietary authoring software and follows all the information requirements needed to be correctly assessed by LiMA Blocks.

CC BY

5.3 Code execution and report

To be correctly assessed, a new verification must be created at the Checking panel. There, the IFC file can be loaded and associated with the regulations that we want to verify. In this case, we associate with the regulation containing the rules created. Then, the next step is to select the created verification and execute it on the compliance check page. After the execution of the verification in the backend, the results of the report appear in the left panel. As shown in Figure 16, the report is capable of highlighting some of the verified elements, depending on the type of verification executed.

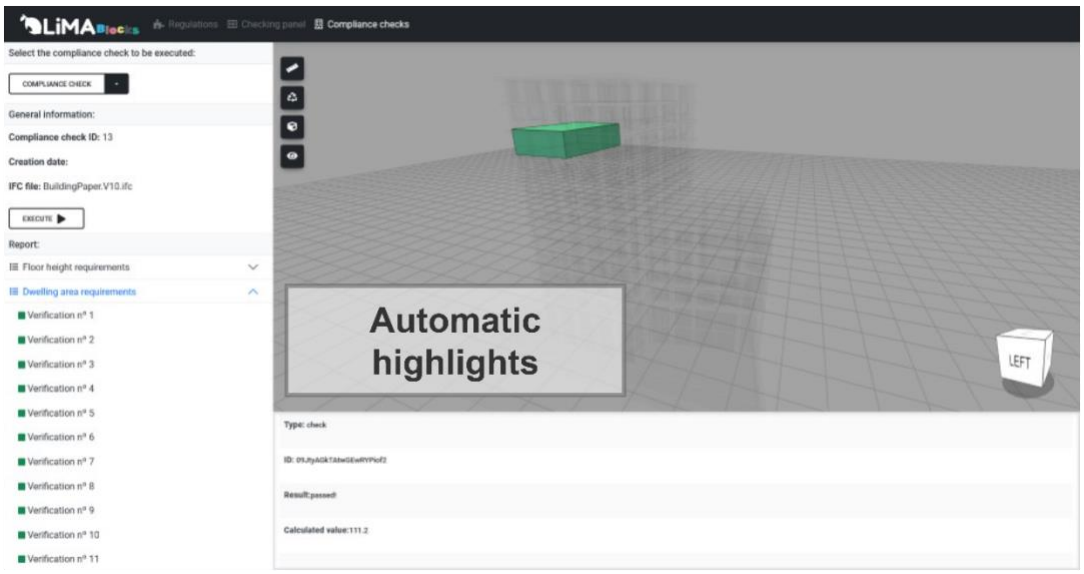

Figure 16: Verification results.

For the first rule, both verifications have failed, since the number of storeys and the height are greater than the one required. As seen in Figure 17, the building can be measured in the viewer with the available measurement tools; for this first rule, however, only the pass/fail outcome is reported, without a recorded numerical comparison.

The results for the second rule were also negative, since the obtained depth of the building was 30.1 m, when the maximum value is 15 m for general buildings (18 m for hotels and public-use facilities). The obtained value was compared with the depth measured in the viewer with its measurement tools, and the two values agreed (Figure 17).

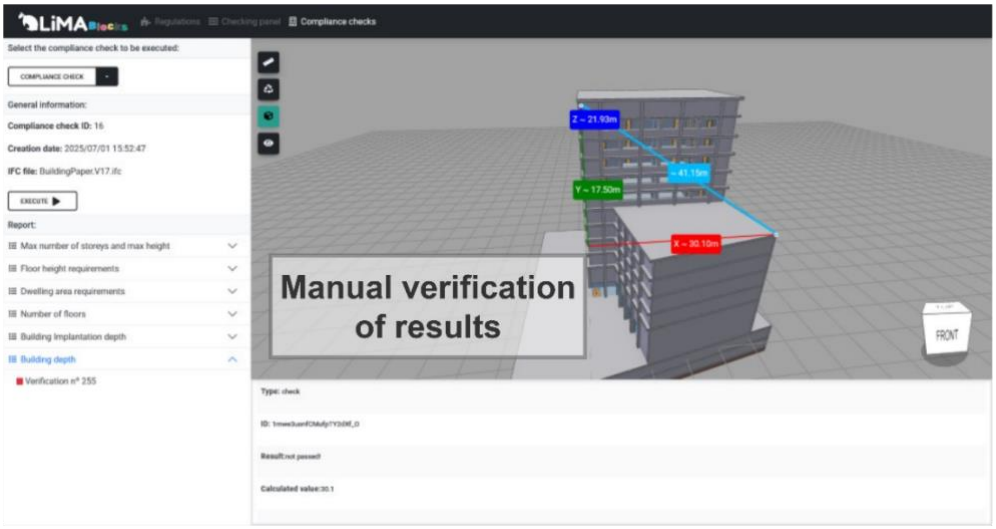

Figure 17: Depth verification results and manual validation through measuring tools.

Finally, the results for dwelling area were positive, since the typology of dwellings of the building has two bedrooms, therefore the minimum area should be 72 m2, while the found area is 111.2 m2. To guarantee the validation of the results obtained by the application, the area was calculated with complementary software, since area measurements are not yet graphically displayed within the LiMA Blocks viewer. The values obtained for the same dwelling in both tools were the same, confirming the accuracy of the verification. Taken together, the results reported in this subsection constitute a verification of execution correctness: they establish that, for the three rules presented and the model used, the outcomes of the checking engine agreed with independent measurement, and that the dwelling area was identical in both tools. This is a statement about the correctness of the execution, and not a measure of the coverage of the regulatory framework or of the scale at which the application has been exercised.

6. DISCUSSION

This section revisits the research question set out in Section 1: How can the lower level of abstraction of programming languages and the IFC schema be bridged to a higher level in a VPL that incorporates local building permit terminology? In response, this paper proposed an open-source approach to the creation of rules and the execution of compliance checks for building permits on IFC models and demonstrated how the lower level of abstraction of programming languages and of the IFC schema can be bridged to a higher level in a visual language. The prototype shows that the components can be integrated end to end: a domain model that maps Portuguese permit concepts to IFC, a block-based language whose blocks generate executable code, and a checking engine that runs the stored rules and reports the results. The approach of storing the rules separately, so that they can be interpreted by the check engine and execute the compliance checks, demonstrates the capability of the application to create machine-readable regulations. Nine representative clauses from three Portuguese regulations were implemented, and for the three presented in section 5 the results of the engine agreed with independent measurement.

The broad set of technology and functions involved in a compliance checking system makes its development a complex task. Works demonstrate that the applicability of compliance checking systems often involves the definition of rules that are specific to a particular domain, yet it makes it even more difficult to implement these systems, since the available compliance checking solutions are closed and can't provide the necessary flexibility to adapt to new rules or concepts. Consequently, one of the paths to implement these specific checks is to develop coding scripts, which demands coding skills, knowledge on the BIM open formats and domain specific knowledge, being a task for whole teams. The use of the VPL rather than formal code is expected to lower the specialisation barrier to be able to create complex rules to check IFC models. This is consistent with the visual programming approaches reviewed in section 2, but the present work differs in being open source and in bridging the language to local permit concepts through a domain model.

In terms of scalability of the system, the comparison with existing solutions depends on where the effort is placed. The preparation of the semantic environment, implementing the domain model, creating the mappings and developing the geometry parsing methods, is an incremental effort, incurred once for each new concept and jurisdiction and then amortised across all the rules that reuse it, whereas in the closed solutions reviewed in Section 2.3 the equivalent effort is recurring and is done per rule. For any verification that is not already shipped as a hardcoded template, the alternative is not the reuse of an existing system but the development of a specific checking script, which demands coding skills, knowledge of the open BIM formats and domain specific knowledge simultaneously. The benefit of the proposed system resides precisely in the dissociating of these two costs: the domain model is expanded only to introduce new concepts, never to introduce new rules, so that the marginal effort of authoring an additional rule is reduced, whenever the required concepts already exist, to a composition of blocks that a permit expert can carry out, instead of code written by a development team. Since regulations are revised considerably more often than the underlying permit concepts are redefined, it is this recurring cost, and not the initial preparation, that governs the long-term feasibility of a compliance checking system within a permit process. This does not eliminate the setup cost, which remains a genuine need of the approach, but it situates it as an investment in a reusable semantic environment rather than as an overhead attached to each individual check.

In terms of the flexibility of the language, compared with the parametric rules and rule templates of closed checkers, the proposed approach relies on the free composition of conditions and checks, as illustrated by the conditional depth limit and the table-based area requirement of section 5.1. This flexibility applies to the

CC BY

composition of rules over concepts that already exist in the domain model; rules that require new concepts still demand development work, as discussed above. For clauses that are not covered by templates, the relevant alternative is custom scripting on open libraries, which offers equivalent expressiveness but requires programming and open BIM knowledge for every rule. Therefore, overall scalability can be considered in this work as a design property rather than as an empirical result: it follows from the architecture presented in Section 4, namely the separation between concepts and rules, the implementation of the modules with open-source technology and the reliance on open formats, each of which is open to inspection. Establishing a complete scalable VPL would require a coverage analysis over a bounded regulatory corpus, classifying each verifiable clause as expressible with the existing blocks, as requiring a new concept in the domain model, or as outside the scope of semantic and geometric checking. Such an analysis over a full corpus was not conducted here and is identified as future work.

A further benefit of the architecture is qualitative: storing the rules separately as inspectable machine-readable pieces, on top of open-source modules and open formats, addresses the transparency requirement that closed and hardcoded implementations cannot satisfy, and which, as discussed above, exposes the public administration to legal risk.

One of the main challenges of this work resided in the attempt to use geometrical processing for verifications, since even the spatial-based concepts, such as depth, within legislation, can be defined using semantic concepts such as balconies. Another example is the area of spaces, that, for a practical implementation, needs to be defined by slicing the volumetric definition of an IfcSpace at a certain height. However, geometric processing can diminish the need to define explicit information of properties and property sets, making the model preparation easier, but also grants coherence between the verified information and the geometry of objects, that can be inspected in viewers.

The domain model was defined based on concepts gathered from clauses of the Portuguese regulatory framework, and this imposed a benefit and a limitation. The level of abstraction proposed for the VPL pushed the role of concept definition and mapping to the domain model, thus, despite having logic blocks that implement general-purpose control structures, such as conditionals and comparisons, borrowed from programming languages, several blocks are simple object-oriented implementations of the permit concepts, making it easy to work with these concepts. On the other hand, these concepts might not be used in other jurisdictions, which implies the need of changing the domain model, being a hurdle on the applicability of this system elsewhere. On the other hand, the choice to make the concepts completely generalized in the scope of building permits would make the system inapplicable, since minor changes in how a height or a distance is calculated imply severe legal problems. Multidisciplinary teams are still necessary to create or expand the domain model, which is a remaining limitation.

7. CONCLUSION

This work presented LiMA Blocks, an open-source web application for building permit compliance checking. Rules are created with a block-based visual language and executed on IFC models through a domain model that maps Portuguese permit concepts to the IFC schema, and a checking engine runs the stored rules and generates the report. Nine clauses from the national regulation (RGEU) and from the municipal plans of Vila Nova de Gaia and Lisbon were implemented as visual rules. They were selected to be representative of the types of requirements found in these documents, namely national and municipal scope, textual and tabular formulation, and values taken from properties, semantic classification and geometry. They are not a sample of the whole regulatory framework. In answer to the research question set out in Section 1, this work shows that the lower level of abstraction of programming languages and the IFC schema can be bridged to a higher level in a VPL that incorporates local building permit terminology by combining a domain model that codifies local permit concepts and maps them to the IFC schema, a block-based VPL whose blocks generate executable code, and a modular architecture whose checking engine runs the stored rules on IFC models, as demonstrated through nine clauses from the Portuguese regulatory framework.

The evidence reported is a technical demonstration. For the three rules presented on a model based on a real design for Vila Nova de Gaia, the outcomes produced by the engine agreed with independent measurement, and the dwelling areas obtained with the engine and with complementary software were identical. This supports the correctness of the execution for those rules and that model. It does not measure regulatory coverage, scale of use or the effort of authoring rules.

CC BY

The main design finding is that separating permit concepts, which are codified once in the domain model, from rules, which are composed from blocks, allows a new rule to be added by composing existing blocks whenever the required concepts exist. This is a property of the design, argued in section 6, and not an empirical result. Rules that require new concepts still demand development work and multidisciplinary teams, and the domain model reflects Portuguese concepts, so the transfer to other jurisdictions is expected from the modular design but was not demonstrated.

Three follow-up studies arise from these limits. First, an evaluation with municipal technicians, to test whether permit experts can create and maintain rules with the visual language. Second, a coverage analysis over a bounded regulatory corpus, classifying each verifiable clause as expressible with the existing blocks, as requiring a new concept in the domain model, or as outside the scope of semantic and geometric checking. Third, the adaptation of the domain model to another jurisdiction, to assess transferability. The contribution of this work is therefore a functional and openly available prototype, and a design on which these studies can be built, that responds to the limitations of hard-coded and closed compliance checking systems.

ACKNOWLEDGEMENT

This work was funded by the Horizon Europe 2021-2027 Research and Innovation Programme through the R&D project Change Toolkit for Digital Building Permit CHEK with reference 101058559. This work was also supported by FCT/MCTES under the R&D Unit Institute for Sustainability and Innovation in Structural Engineering (ISISE), under the references UID/4029/2025 (https://doi.org/10.54499/UID/04029/2025), UID/PRR/04029/2025 (https://doi.org/10.54499/UID/PRR/04029/2025) and UID/PRR2/04029/2025 (https://doi.org/10.54499/UID/PRR2/04029/2025), and under the Associate Laboratory Advanced Production and Intelligent Systems ARISE under reference LA/P/0112/2020 (https://doi.org/10.54499/LA/P/0112/2020).

REFERENCES

Aida, J., Bitencourt, B., & Checcucci, É. (2022). Análise da plataforma de verificação de regras de salvador, a metropolis. In B. Figueiredo, J. P. Martins, J. Granja, J. C. Lino, & M. Azenha, *4º congresso português de 'building information modelling' vol. 2—ptBIM* (1st edn, pp. 178–187). UMinho Editora. https://doi.org/10.21814/uminho.ed.77.15

Amor, R., & Dimyadi, J. (2021). The promise of automated compliance checking. *Developments in the Built Environment*, *5*, 100039. https://doi.org/10.1016/j.dibe.2020.100039

Arnold. (2022, September 4). Meet RuleML. *Ruleml*. https://www.ruleml.org/meet-ruleml/

Daum, S., & Borrmann, A. (2015). *Simplifying the Analysis of Building Information Models Using tQL4BIM and vQL4BIM*.

Dimyadi, J. (2016). *Integrating the BIM Rule Language into Compliant Design Audit Processes*.

Eastman, C., & Henrion, M. (1977). Glide: A language for design information systems. *SIGGRAPH Comput. Graph.*, *11*(2), 24–33. https://doi.org/10.1145/965141.563863

Eastman, C., Lee, J., Jeong, Y., & Lee, J. (2009). Automatic rule-based checking of building designs. *Automation in Construction*, *18*(8), 1011–1033. https://doi.org/10.1016/j.autcon.2009.07.002

Eaton, J. (2021). *Rules as Code: How Technology May change the Language in which Legislation is Written, and What it Might Mean for Lawyers of Tomorrow*.

Fauth, J., Nørkjær Gade, P., Kaiser, S., Raj, K., Goul Pedersen, J., Olsson, P.-O., Nisbet, N., Mastrolembo Ventura, S., Hirvensalo, A., Granja, J., Urban, H., Rutešić, S., Verstraeten, R., Raitviir, C.-R., Kallinen, A.-R., Schranz, C., Stojanov, T., & Tekavec, J. (2025). Investigating building permit processes across europe: Characteristics and patterns. *Building Research & Information*, *53*(4), 417–434. https://doi.org/10.1080/09613218.2024.2400467

Fenves, S. (1966). Tabular Decision Logic for Structural Design. *Journal of the Structural Division 92*, 473–490.

Fenves, S. J., Wright, R. N., Stahl, F. I., & Reed, K. A. (1987). *Introduction to SASE: Standards analysis, synthesis, and expression* (NBS IR 87-3513; 0 edn, p. NBS IR 87-3513). National Bureau of Standards. https://doi.org/10.6028/NBS.IR.87-3513

Ghannad, P., Lee, Y.-C., Dimyadi, J., & Solihin, W. (2019). Automated BIM data validation integrating open-standard schema with visual programming language. *Advanced Engineering Informatics*, *40*, 14–28. https://doi.org/10.1016/j.aei.2019.01.006

Hjelseth, E., & Nisbet, N. N. (2011). *Capturing normative constraints by use of the semantic mark-up RASE methodology*. https://www.semanticscholar.org/paper/Capturing-normative-constraints-by-use-of-the-RASE-Hjelseth-Nisbet/8820d782f37a3bccc5922dc5e6ee1b344a027828

Kim, H., Lee, J.-K., Shin, J., & Choi, J. (2019). Visual language approach to representing KBimCode-based korea building code sentences for automated rule checking. *Journal of Computational Design and Engineering*, *6*(2), 143–148. https://doi.org/10.1016/j.jcde.2018.08.002

Ledoux, H., Arroyo Ohori, K., Kumar, K., Dukai, B., Labetski, A., & Vitalis, S. (2019). CityJSON: A compact and easy-to-use encoding of the CityGML data model. *Open Geospatial Data, Software and Standards*, *4*(1), Article 1. https://doi.org/10.1186/s40965-019-0064-0

Lee, J. K. (2011). *Building environment rule and analysis (BERA) language and its application for evaluating building circulation and spatial program* [Georgia Institute of Technology]. https://smartech.gatech.edu/handle/1853/39482

Li, L., & Chou, W. (2011). Design and Describe REST API without Violating REST: A Petri Net Based Approach. *2011 IEEE International Conference on Web Services*, 508–515. https://doi.org/10.1109/ICWS.2011.54

Mateus, R. J. G., Pinto, F. S., Fauth, J., Azenha, M., Granja, J., Veludo, R., Muniz, B., Reis, J., & Marques, P. (2024). Improving regulations for automated design checking through decision analysis good practices: A conceptual application to the construction sector. In Silva F.J.G., Campilho R.D.S.G., & Pereira A.B. (Eds), *Lect. Notes Mech. Eng.* (pp. 160–169). Springer Science and Business Media Deutschland GmbH. Scopus. https://doi.org/10.1007/978-3-031-38241-3_19

Mazairac, W., & Beetz, J. (2013). BIMQL – An open query language for building information models. *Advanced Engineering Informatics*, *27*(4), 444–456. https://doi.org/10.1016/j.aei.2013.06.001

Morris, J. (2020). Blawx: Rules as Code Demonstration. *MIT Computational Law Report*.

Nawari, N. O. (2019). A Generalized Adaptive Framework (GAF) for automating code compliance checking. *Buildings*, *9*(4). https://doi.org/10.3390/buildings9040086

Noardo, F., Guler, D., Fauth, J., Malacarne, G., Mastrolembo Ventura, S., Azenha, M., Olsson, P.-O., & Senger, L. (2022). Unveiling the actual progress of digital building permit: Getting awareness through a critical state of the art review. *Building and Environment*, *213*, 108854. https://doi.org/10.1016/j.buildenv.2022.108854

Palmirani, M., & Vitali, F. (2011). Akoma-ntoso for legal documents. In G. Sartor, M. Palmirani, E. Francesconi, & M. A. Biasiotti (Eds), *Legislative XML for the Semantic Web: Principles, Models, Standards for Document Management* (pp. 75–100). Springer Netherlands. https://doi.org/10.1007/978-94-007-1887-6_6

Park, S., & Lee, J.-K. (2016, July 21). *KBimCode-Based Applications for the Representation, Definition and Evaluation of Building Permit Rules*. 33th International Symposium on Automation and Robotics in Construction. https://doi.org/10.22260/ISARC2016/0087

Park, S., Lee, Y.-C., & Lee, J.-K. (2016). Definition of a domain-specific language for korean building act sentences as an explicit computable form. *Journal of Information Technology in Construction*, *21*, 422–433. Scopus.

Pasternak, E., Fenichel, R., & Marshall, A. N. (2017). Tips for creating a block language with blockly. *2017 IEEE Blocks and Beyond Workshop (B&B)*, 21–24. https://doi.org/10.1109/BLOCKS.2017.8120404

Pedro, J. B., Meijer, F., & Visscher, H. (2011). *Comparison of building permit procedures in European in European Union countries Cova da Moura Rehabilitation Needs Assessment View project Building Regulations for*

Construction Works in Existing Buildings View project Comparison of building permit procedures in European Union countries. https://www.researchgate.net/publication/257527312

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A design science research methodology for information systems research. *Journal of Management Information Systems*, *24*(3), 45–77. https://doi.org/10.2753/MIS0742-1222240302

Preidel, C., & Borrmann, A. (2015). *Automated code compliance checking based on a visual language and building information modeling*. 1–8. https://doi.org/10.22260/ISARC2015/0033

Preidel, C., Daum, S., & Borrmann, A. (2017). Data retrieval from building information models based on visual programming. *Visualization in Engineering*, *5*(1), 18–31. https://doi.org/10.1186/s40327-017-0055-0

Regime Jurídico Dos Instrumentos de Gestão Territorial - RJIGT | DR, Decree-law 80/2015 (2015). https://diariodarepublica.pt/dr/legislacao-consolidada/decreto-lei/2015-105297982

Santos, M. F. de S. (2021). *Metodologias BIM para verificação regulamentar em contexto de licenciamento municipal: Proposta, implementação e aplicação* [masterThesis]. https://repositorium.sdum.uminho.pt/handle/1822/76900

Santos, M., Vieira, L. L., Granja, J. L. D., Magalhães, T. P., Mateus, R., Carvalho, M., & Azenha, M. (2022). *Automatização parcial da verificação regulamentar de projetos de edificação para efeitos de licenciamento urbanístico: Proposta, implementação e aplicação piloto*. https://doi.org/10.21814/uminho.ed.77.20

Schranz, C., Urban, H., Fischer, S., Eichler, C. C., Krischmann, T., & Hopferwieser, M. (with Tu Wien). (2026). *BIMcert handbook: Basic knowledge openBIM*. Mironde-Verlag. https://doi.org/10.34726/11822

Seraj, M., Katterfeldt, E.-S., Bub, K., Autexier, S., & Drechsler, R. (2019). Scratch and Google blockly: How girls' programming skills and attitudes are influenced. *Proceedings of the 19th Koli Calling International Conference on Computing Education Research*, 1–10. https://doi.org/10.1145/3364510.3364515

Soliman-Junior, J., Tzortzopoulos, P., Baldauf, J. P., Pedo, B., Kagioglou, M., Formoso, C. T., & Humphreys, J. (2021). Automated compliance checking in healthcare building design. *Automation in Construction*, *129*, 103822. https://doi.org/10.1016/j.autcon.2021.103822

Tan, X., Hammad, A., & Fazio, P. (2010). Automated Code Compliance Checking for Building Envelope Design. *Journal of Computing in Civil Engineering*, *24*(2), 203–211. https://doi.org/10.1061/(ASCE)0887-3801(2010)24:2(203)

van Berlo, L., Krijnen, T., Tauscher, H., Liebich, T., van Kranenburg, A., Paasiala, P., & Paasiala, P. (2021). *Future of the industry foundation classes: Towards IFC 5*.

Vom Brocke, J., Hevner, A., & Maedche, A. (2020). Introduction to design science research. In J. Vom Brocke, A. Hevner, & A. Maedche (Eds), *Design science research. Cases* (pp. 1–13). Springer International Publishing. https://doi.org/10.1007/978-3-030-46781-4_1

Waddington, M. (2020). Rules as Code. *Law in Context*, *37*(1), 179–186. https://doi.org/10.26826/law-in-context.v37i1.134

Witt, A., Huggins, A., Governatori, G., & Buckley, J. (2024). Encoding legislation: A methodology for enhancing technical validation, legal alignment and interdisciplinarity. *Artificial Intelligence and Law*, *32*(2), 293–324. Scopus. https://doi.org/10.1007/s10506-023-09350-1

Wülfing, A., Windisch, R., & Scherer, R. J. (2014). A visual BIM query language. *Researchgate*. https://doi.org/10.13140/2.1.2158.9769

Yang, G., Kifer, M., & Zhao, C. (2003). Flora-2: A Rule-Based Knowledge Representation and Inference Infrastructure for the Semantic Web. In R. Meersman, Z. Tari, & D. C. Schmidt (Eds), *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE* (Vol. 2888, pp. 671–688). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-39964-3_43

Zhang, J., & El-Gohary, N. M. (2017). Integrating semantic NLP and logic reasoning into a unified system for fully-automated code checking. *Automation in Construction*, *73*, 45–57. https://doi.org/10.1016/j.autcon.2016.08.027

Zijing, Z., Zhang, Z., Nisbet, N., Nisbet, N., Ling, M., Ma, L., Broyd, T., & Broyd, T. (2023). Capabilities of rule representations for automated compliance checking in healthcare buildings. *Automation in Construction*. https://doi.org/10.1016/j.autcon.2022.104688

CC BY