The European Master in Building Information Modelling is a joint initiative of:

Univerza *v Ljubljani*

Universidade do Minho

POLITECNICO MILANO 1863

Design and Fabrication of Farming Greenhouses

Tijana Miljic

BIM A+

UMinho | 2023

Universidade do Minho
Escola de Engenharia

Tijana Miljic

Design and Fabrication of Farming Greenhouses

BIM A+ European Master in Building Information Modelling

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

October, 2023

Universidade do Minho
Escola de Engenharia

Tijana Miljic

Design and fabrication of farming greenhouses

European Master in
Building Information Modelling

Master Dissertation
European Master in Building Information Modelling

Work conducted under supervision of:
Prof. Isabel Valente
Eng. Vitor Cruz (tutor in the company)

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

October, 2023

ACKNOWLEDGMENTS

I would like to express my gratitude to the Erasmus Joint Master Degree committee for awarding me with a full scholarship that allowed me to be part of this invaluable experience, which has been a crucial step in my career.

I want to extend my thanks to the professors and staff at BIM A+ for tailoring and making this master program possible and providing all the valuable lessons that have inspired me to continue learning and enhancing my knowledge.

Thank you, to my mentor Prof. Isabel Valente, and engineers from the company Estufasminho Eng Vitor Cruz and Luís Oliveira for providing me with information and tailoring this interesting and up-to-date topic.

Thank you, Nikola Petkovic, for showing genuine interest in my work, for the interesting and informative discussions, support, and insightful suggestions throughout this journey.

Thank you, my family, for your constant emotional support motivation, and inspiration.

STATEMENT OF INTEGRITY

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

[illegible]

RESUMO

Esta dissertação de mestrado explora a automatização de tarefas repetitivas no projeto de estruturas modulares de estufas e o trabalho é desenvolvido com base na utilização do Autodesk Inventor e do iLogic. As estufas são construções essenciais ao cultivo agrícola, permitindo que este se desenvolva durante todo o ano. Normalmente, apresentam uma gama de variações geométricas, tais como estruturas de um ou vários vãos, juntamente com diferentes configurações de túneis. A modularidade destas estruturas possibilita a automatização do seu projeto através da geração automática de elementos ou partes da estrutura.

De modo a compreender a estrutura da estufa, os elementos constituintes e as sua variações foram sistematicamente analisados e organizados em diagramas. Estes diagramas serviram de base ao desenvolvimento subsequente de uma regra externa no iLogic e no Inventor. O principal veículo para esta automatização foi o iLogic, que facilitou o acesso à API do Inventor, permitindo assim a incorporação das funcionalidades do Inventor no código desenvolvido.

O resultado deste estudo é uma regra externa, que com base nos inputs do utilizador, gera uma estrutura de estufa em segundos, proporcionando uma maior eficiência no processo de desenho. Este trabalho contribui para expandir o conhecimento associado à geração automática de estruturas modulares complexas com múltiplos parâmetros, como é o caso das estufas agrícolas.

Palavras chave: (Geração automatizada de montagens, iLogic, Inventor, Estrutura modular, VB.NET)

ABSTRACT

This master's dissertation explores the automation of repetitive tasks in the design of modular greenhouse structures, with a focus on utilizing Autodesk Inventor and iLogic. Greenhouses, essential for year-round crop cultivation, exhibit a range of geometrical variations, such as single-span and multi-span structures, along with differing tunnel configurations. The modularity of these structures opens the possibilities for automation of the design process through automation of assembly generation.

To understand greenhouse structure, constituent elements, and element variations were systematically analyzed and organized into diagrams. These diagrams served as the foundation for the subsequent development of the external rule within iLogic and Inventor. The primary vehicle for this automation was iLogic, which facilitated access to the Inventor API, thereby enabling the incorporation of Inventor's functionalities into the code.

The outcome of this study is an external rule, that based on user inputs, generates a greenhouse structure within seconds providing better efficiency in the design process. This work aims to contribute to the knowledge about automated assembly generations of complex modular structures with multiple parameters such as greenhouses.

Keywords: (Automated assembly generation, iLogic, Invenotr, Modular structure, VB.NET)

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

In an era of rapid population growth, challenges of providing food are imposed on the agricultural sector more than ever. Greenhouses present an important source of food and provide a controlled environment that allows for year-round cultivation of crops, regardless of the external weather conditions. These structures are modular complex systems used to protect crops from wind, rain, snow, and hail, as well as from disease and extreme temperatures. By promoting good conditions for crop growth and seeking to optimize the transmission of solar radiation, humidity, temperature, and CO2 levels, these structures play a key role in optimized farming practices and maximization of crop yields.

The productivity of the construction industry is marced among the lowest compared to other industries (Changali, Mohammad and Van Nieuwland, 2015). The focus of this work is the design phase of greenhouse structures, where its modularity allows for automation of the design process which presents improvement in productivity and efficiency.

Estufasminho, a Portuguese company specialized in the manufacture and installation of greenhouses, possesses extensive marcet expertise. Despite its established presence, the company aspires to leverage modern technologies to enhance its design and fabrication procedures through the automation of repetitive tasks.

The subject of this research is the implementation of parametric design and the introduction of new methodologies and workflows into the design process of greenhouses. The work that is developed relies on collaboration with the company Estufasminho. The company has been using Autodesk Inventor as an authoring tool for many years and is now looking to explore its capabilities further. During this time extensive library of models was created, that will be used in this work. Therefore, the option of using other software was not considered in this thesis. By analyzing the design process, structure, modules, and their variations, this thesis will give a solution on how the integration of adequate tools, like Inventor API and iLogic, can automate tasks, reduce manual work, and increase overall productivity.

The recognized problem at hand is the inefficiency inherent in manual design processes of modular construction, leading to prolonged design cycles and increased labor demands. This research seeks to address this problem by exploring the possibilities of greenhouse parametrization, investigating how to implement parametric design, through Inventor API and iLogic to significantly reduce manual work, expedite design phases, and enhance overall productivity.

These are the hypotheses considered for the work:

1. The integration of parametric design methodologies, with the assistance of Inventor API and iLogic, will lead to a reduction in the time required to generate a 3D greenhouse model;

2. Automated design processes will result in a decrease in human error, leading to enhanced accuracy and consistency in greenhouse design;

3. The exploration of greenhouse parametrization through the utilization of Inventor API and iLogic for greenhouse design will allow designers to focus more on critical decision-making aspects, ultimately contributing to higher value-added tasks.

The specific objectives guiding this research are as follows:

1. To analyze the greenhouse structure, its elements, and its variations.

2. To explore how user inputs influence the choice of elements and how parameters influence each other.

3. To explore the applicability of parametric design in the context of modular greenhouse structures and how to support element variations and different greenhouse configurations for greenhouse structures, using parametric design.

4. To analyze the process of manual creation of assembly and explore capabilities of Inventor API and iLogic in automating repetitive tasks within the design.

5. With the utilization of Inventor API and iLogic, to develop a system that automates the assembly generation of greenhouse structures.

This thesis aims to give answers on how Inventor API and iLogic can be used as tools for the automatization of greenhouse model generation. By giving answers to this question study aims to contribute to more sustainable practices in greenhouse design.

The structure of the thesis is divided into six chapters.

Chapter 2 *State of the art* presents a literature review organized around key themes crucial to this work. It is divided into subchapters. The first subchapter (Greenhouse typology) explains greenhouse structure, the main influences on its design, classifications, and structure with elements included in the case study. The second subchapter (Automation of the design process) presents ideas from the manufacturing industry that can be implemented in modular construction. These terms are modularization and modules, the configuration of the product, and how functionalities chosen by the customer define the product. The following subchapter explains methodologies used to capture and organize domain knowledge that will be used in the creation of the software solution for product configuration. At the end of this chapter basic principles of parametric design are explained.

Chapter 3 *Methodology* is divided into 3 subchapters. The first one (Needs analysis) elaborates on the need for the tool development. The process of the design phase is analyzed through the AS IS diagram and improvements to that process are suggested. The second subchapter (Choice of tools) closely introduces used tools, Autodesk Inventor, authoring software used at company Estufasminho.

The last subchapter (Domain knowledge systematization) collects and structures domain knowledge for easier implementation into the code.

Chapter 4 *Software design* is organized into seven subchapters. The first subchapter, (Software Structure), elaborates on the components and relationships within software structure. Subsequent subchapters, such as "Input Validation," provide insights into inputs and data types to ensure task success. "Naming Conventions" in the next section details the establishment of consistent naming conventions throughout the development process. Furthermore, the "Rectangular Pattern Command," is explored as a crucial element used extensively throughout the process. Moving forward, the following subchapter explains the code and its design, highlighting the commands employed and the underlying logic. After that, it is explored how to incorporate new elements into the greenhouse design. This chapter is followed by a use-case scenario. The final subchapter explains the code responsible for generating structures with multiple tunnel widths.

Chapter 5 *Discussion* elaborates on findings and solutions that are the answers to the objectives, set for this research. The subchapter “Code outputs” shows the geometry that is generated as the output of the code. In the following subchapters, the Implementations, and Limitations of the research are elaborated.

Chapter 6 *Conclusion* concludes this work with concise findings and suggestions for future work.

This page is intentionally left blank

2. STATE OF THE ART

2.1. GREENHOUSE TYPOLOGY

Greenhouses are constructions that provide optimal conditions for crop growth by controlling climate elements and providing protection. They are required to allow high light transmittance, low heat consumption, sufficient ventilation efficiency, adequate structural strength and good overall mechanical behavior, low construction, and operating costs. (B Von Elsner *et al.*, 2000). These structures range in size from small sheds to industrial-sized buildings and usually present modular configurations. Each farm is implanted in a land with a specific area and topography, making it essential to adjust the dimensions and disposition of the greenhouses to the type of plantations and plots. They are also filled with equipment, including screening installations, heating, cooling, and lighting, and may be controlled by a computer to optimize conditions for plant growth. Factors that influence the design of greenhouse structures are diverse, however, according to the literature dominant ones are:

1. Climate and latitude of the location,
2. Load requirements depending on the climatic conditions,
3. Local building regulations,
4. Cultivated crops,
5. Availability of building materials.

Greenhouses can be classified concerning different criteria, including structure, purpose, technology, control systems used, covering materials (glass, rigid plastics, plastic, or combinations of these), and construction materials (steel, aluminium, wood, or combinations of these). etc. The one with more interest for this work is a classification according to construction characteristics (width, single or multi-span, sidewall height, roof shape, and slope), (B Von Elsner *et al.*, 2000). Greenhouse structures should be designed according to climate conditions, and general design criteria as well as locally available and cost-effective materials. As per (Christian von Zabeltitz, 2011), the following factors are important in a successful greenhouse design: shape, orientation, structure, cladding material, foundation, ventilation, and technical equipment for climate control.

Figure 1 presents the most frequent greenhouse shapes according to Zabelitz. Saddle roof (a), saw tooth or shed roof (b), round-arched tunnel (c), round arch with a vertical side wall (d), pointed arch with a sloping side wall (e), and pointed arch with a vertical side wall (f). Preferable shapes for covering with plastic film are (e) and (f). Each one of the recognized types has different characteristics and benefits.

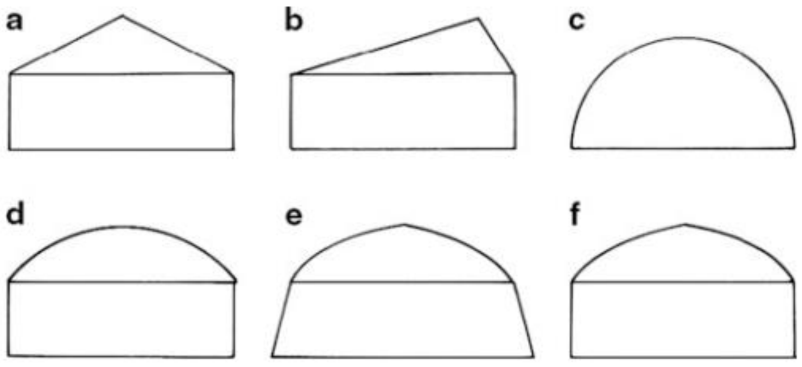


Figure 1 – Most frequent shapes of greenhouses (Christian von Zabeltitz, 2011)

In developing countries with mild climates, the most utilized structure is the *round-arched tunnel* greenhouse. If the construction is dimensioned properly the advantages of this structure are its simplicity and high wind resistance. However, some disadvantages are not to be ignored. In terms of area utilization, the presence of a zone with insufficient height next to the greenhouse sides leads to a decrease in floor usability. Moreover, the rounded roof design results in the creation of a condensation zone, which may potentially harm the crops when dripping occurs. This structure can be improved by introducing vertical walls into a round tunnel greenhouse, resulting in better plot usability that is favorable for plant growth. Additionally, pointed-arched construction offers the advantage of directing drops down the covering film, preventing the formation of a dripping zone. They also reduce the snow load because snow can slide more easily down the roof to the ground or the gutter. However, the mechanical resistance of the Gothic arch is lower than one with a normal arch due to curvature discontinuity, (B. Von Elsner *et al.*, 2000). Figure 2 shows the mentioned types of greenhouse structures.

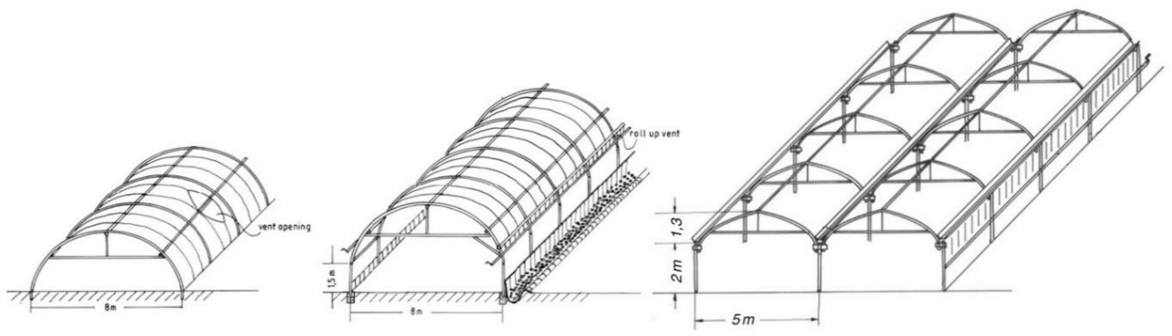


Figure 2 – Round-arched greenhouses, Round-arched greenhouses with vertical walls, and Pointed-arched greenhouses (Gothic arch) (Christian von Zabeltitz, 2011)

Click or tap here to enter text.

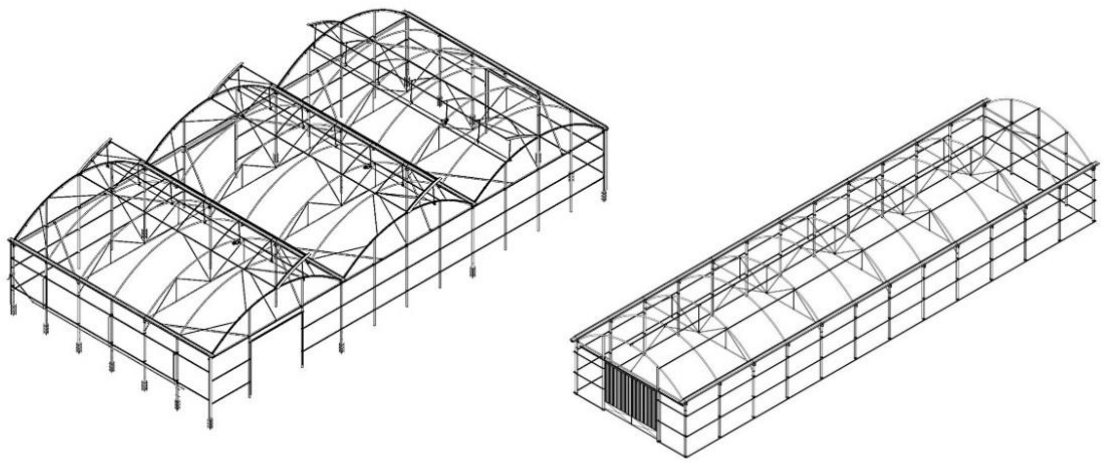


Figure 3 – Multispan and single-span greenhouse construction (Estufasminho, no date)

According to the number of tunnels, greenhouse construction can be single-span and multi-span (Figure 3). Multi-span greenhouses have numerous advantages. They offer larger volumes and improved climatic conditions. It is beneficial to have sidewalls as high as possible, preferably around 3m, while also ensuring wind resistance. Efficient ventilation is achieved through a combination of sidewall and gable ventilators, with the possibility of using mechanical ventilators. The usage of the plot under these structures is maximized due to vertical sidewalls, (Christian von Zabeltitz, 2011).

Greenhouse structure consists of numerous elements. The European standard for greenhouses EN 13031-1 (2001) specifies design and construction requirements for the mechanical resistance, stability, serviceability, and durability of commercial greenhouse structures. Metallic components are connected using clamps or screws (Figure 4), with notice that welding should be avoided after galvanization. Column capitals support the roof structure and gutters. Gutters guarantee efficient drainage of rainwater. Moreover, preventing water penetration from the side walls, gable, and roof into the greenhouse is vital. Ventilation can be achieved either through the ridge or on the side walls. Other important elements are the foundations, where instructions and possible tolerances are closely defined. Steel profiles (negativo) are utilized to link the foundation with columns that are fastened on it.

Estufasminho, the collaborative partner for this case study, produces two types of greenhouses: the Round Arch with a vertical sidewall, and the Pointed Arch with a vertical sidewall.

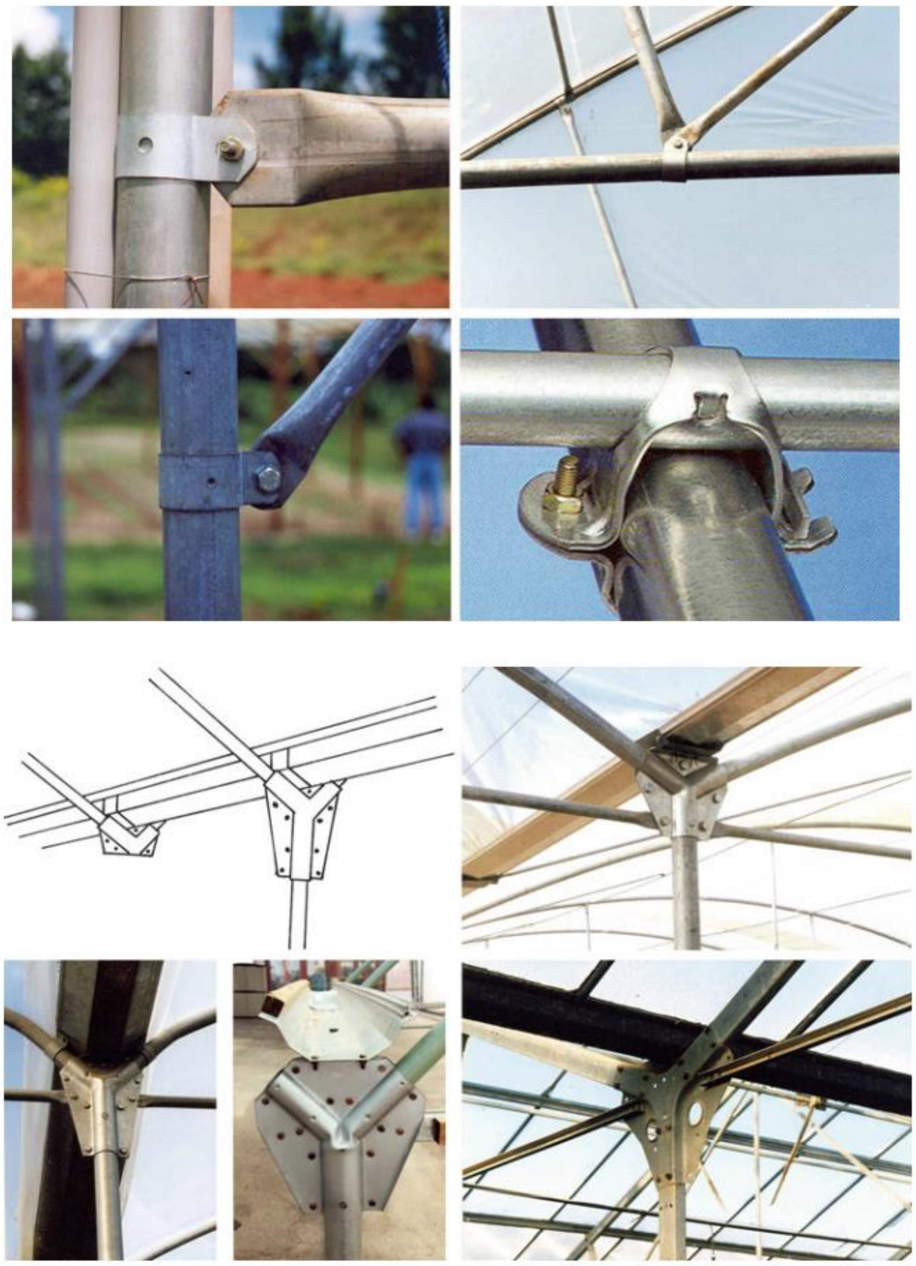


Figure 4 – Clamps for steel components, Connectors for gutters and roof pipes (capitel) (Christian von Zabeltitz, 2011)

2.2. AUTOMATION OF DESIGN PROCESS

2.2.1. The current state of productivity in the Construction industry

By employing seven percent of the world's working population, the construction industry holds a significant position in the global economy. Despite its substantial workforce, the productivity of the sector is among the lowest, with labor-productivity growth of only 1 percent, compared with the growth of 2.8 percent for the total world economy (Agarwal, Chandrasekaran and Sridhar, 2016). Consequently, low productivity in the sector leads to deadline extensions and significant budget overruns. According to McKinsey research, 98 percent of megaprojects suffer cost overruns of more than 30 percent, while 77 percent are at least 40 percent late.

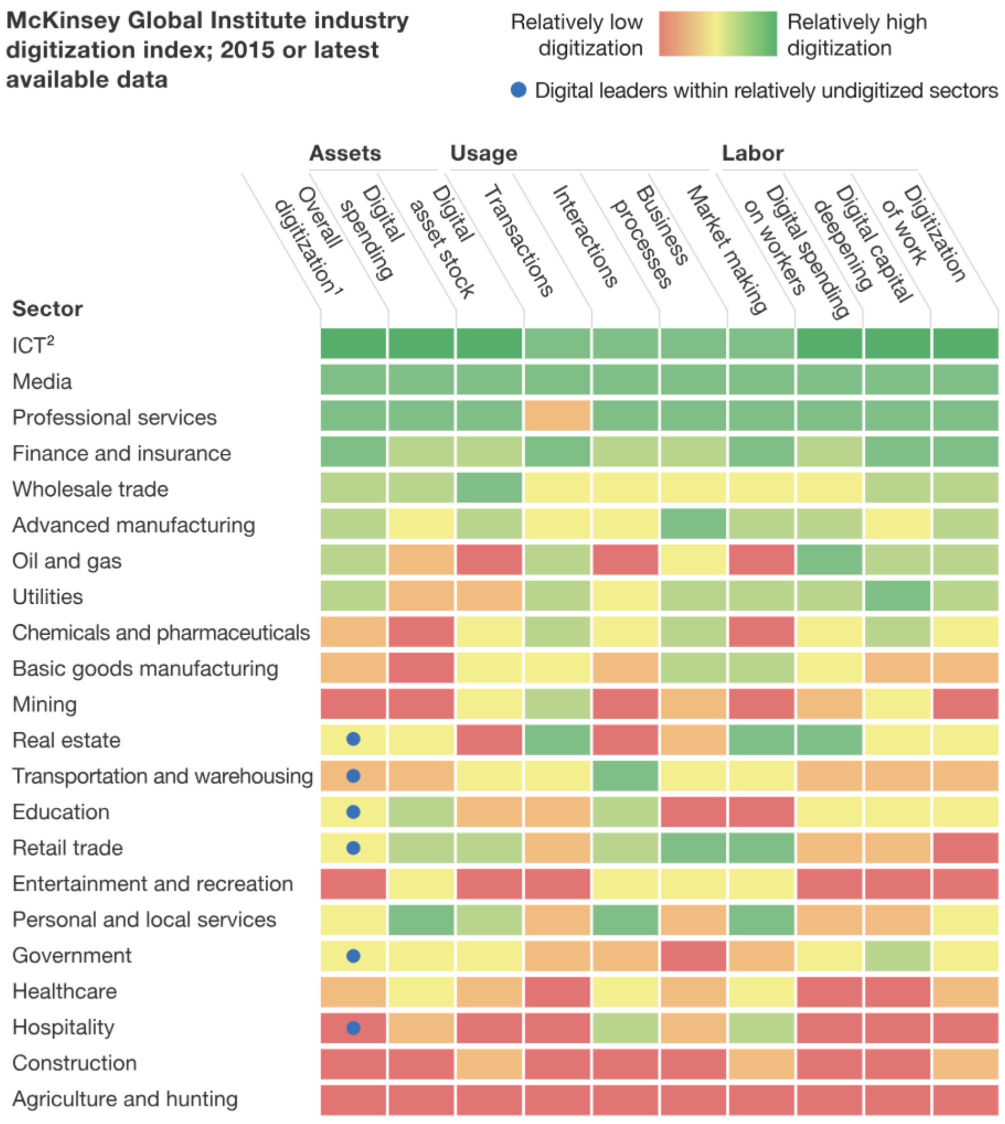


Figure 5 – Index of digitalization among industries (Agarwal, Chandrasekaran and Sridhar, 2016)

As per McKinsey's research (Figure 5), the construction industry is among the least digitized across various segments. Examples from previously mentioned research show that investment in Research and Development is less than 1% of revenues compared to 3.5% to 4.5% in the auto and aerospace sectors. Similarly, spending on information technology (IT) is also minimal in construction, despite the availability of new software solutions designed for the industry. Low Digitalisation is one of the segments which if improved could significantly impact productivity up to 14 to 15 percent and cost reduction of 4 to 6 percent. (Agarwal, Chandrasekaran and Sridhar, 2016)

For a better understanding of digital construction, two terms can be distinguished. The first one is called *Digitisation*, where information is converted into digital form, and the Second one is *Digitalisation* - where business is optimized with digital information (from sensors, drones, photos, and construction drawings). (ING, 2022)

Digital collaboration and mobility, 5D building information modeling, the Internet of Things, and advanced analytics are among many ideas and trends that are emerging and helping in the digitalization of the construction industry. However *Parametric design* and *Configure to order platforms* are the ones that are of importance for this study among many other efficient applications. The parametric design will be elaborated in a separate chapter.

Configure-to-order platforms offer clients the possibility to tailor products according to their own needs and to submit an order without the intervention of an architect. (ING, 2022) An example of this production process is IKEA Kitchen Planner where clients, online, can customize their kitchen with real-time visualization, pricing, and order placement. The interface of this platform, presented in Figure 6, is intuitive and user-friendly with all necessary information visible. These platforms, whether designed for direct customer use or utilized by engineers, offer solutions to automate repetitive tasks, during the design phase when construction is made of modular elements.

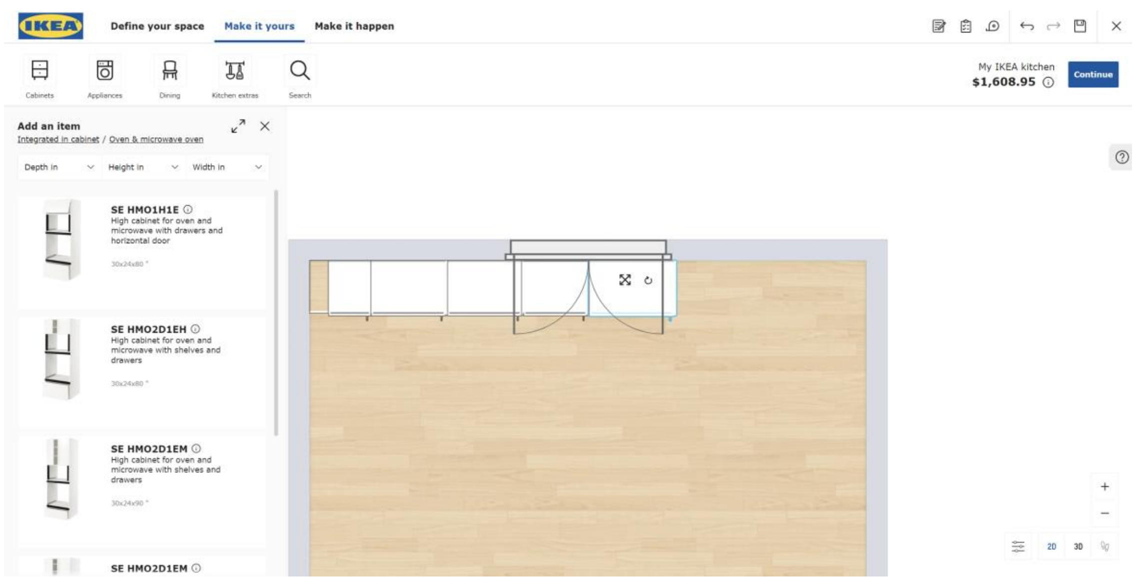


Figure 6 – Example of configure-to-order platform (Source:https://kitchen.planner.ikea.com/)

IBM defines automation as the ”use of technology to perform tasks where human input is minimized.” (IBM, no date). In his work “Design automation in construction,” Sandberg recognizes factors that improve automation. Among others, the important concepts that are implemented in this case study are *Knowledge–based engineering,* and *Modules and Modularizations*. (Sandberg *et al.*, 2016)

Having in mind that the construction industry has to adapt to given conditions, automation will never be complete. However, the construction industry could take knowledge of modularity and process customization from the manufacturing industries (Barbosa Filipe *et al.*, 2017). Additionally, automation in the design process can help in reusing successful solutions, generation of multiple options, tracking and correcting modifications, and increasing responsiveness toward the client, which is an important service, in a fast fast-paced business environment.

2.2.2. Modularization

Modularization is defined as the decomposition of a product into modules (Ericsson and Erixon, 1999). In the Oxford dictionary, the module is defined as a standardized part or independent unit that can be used to construct a more complex structure. Greenhouse structures, fabricated by the company Estufasminho, include standardized parts that are further used to develop different variants, capable of adapting to different terrains and plots. Three types of modules can be recognized: the *Standardized module* is present in all designs, the *Variant module* has to be configured to fit the project, and the *Design module* needs to be uniquely designed to fit a specific project, (Jensen *et al.*, 2014). In the construction industry, modularity was successfully integrated through the idea of “Cut to fit” modularity. This concept involves parametrization, as the module’s interface remains consistent while allowing a change of dimensions. Essentially, the module can be adapted and adjusted as needed without altering its fundamental structure, offering flexibility and efficiency in the construction process, (Jensen, Hamon, and Olofsson, 2009).

According to Jørgensen, the main advantage of modularization is that the end product can vary in shape and dimensions, but the design and production of components and modules within a product family are the same. The design phase is replaced by a configuration phase where the product is customized by selecting functionalities that define which module variants will become part of the product. (Jørgensen K. A., 2001)

According to Smiding a methodology for developing modular platforms within construction includes the following phases.

1. The process starts with analyzing various project blueprints to capture and evaluate marcet needs, and requirements and identifying variations in technical solutions.
2. Generic product architecture refers to a fundamental, high-level design structure that is common to all products within a certain category or domain. In this step it is decomposed into modules and parts, categorizing the suggested technical solutions accordingly. For example in the case of a greenhouse structure can be decomposed into an arc structure, foundation options, covering material, ventilation structure, etc.

3. For each module, different technical solutions are analyzed to assess whether they can satisfy the identified requirements. For example, for covering material it can be considered polycarbonate, glass, plastic film, etc.
4. The following step focuses on developing and determining the flexibility, capabilities, and constraints of each module and part.
5. Finally, once the functional requirements, product architecture, and product flexibility are defined and controlled, the process of creating the product configurator can start. (Jensen *et al.*, 2014) A product configurator presents a tool or a software application that enables customers or users to customize and design a specific product according to their preferences and requirements. It provides an interactive and user-friendly interface through which customers can choose various features, options, and specifications of a product to create a personalized version that meets their needs.

2.2.3. Configuration of a product

This work aims to implement some of the manufacturing industry concepts into the construction process. For better systematization of products, efficient communication and minimization of misunderstandings are necessary which are achieved through clear and well-defined terminology and systematic methodology.

The manufacturing industry adopts perfected methodologies and principles to attain high productivity. Greenhouse structures consist of modular elements, like many other products in the manufacturing industry. Certain terms related to the principles of the manufacturing industry need clarification before being implemented. Understanding what product configuration is, will help to choose and describe modules that will be implemented later. Moreover, the implementation of new elements or modules is based on pre-established procedures.

Product configuration is a process in which the customer chooses from a set of options to create a product that meets individual requirements. At its simplest, product configuration involves choosing the suitable modules. In other words, the *Product family* can be looked at as a template, a foundation, with a set of open specifications, where the product is defined through the configuration process, (Jørgensen K. A., 2001). In the context of the construction industry product families should have a similar product structure, like multistorey buildings, single-family houses, concrete bridges, or in this study greenhouse. Figure 7 explains the process of configuration, where the configured product is derived from the broader product family, followed by the next step in which the physical model is developed.

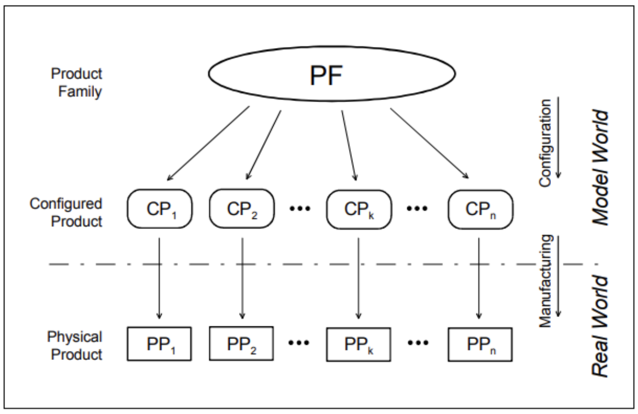


Figure 7 – The product family model as the foundation for product configuration (Jørgensen K. A., 2001)

The product consists of several modules and modules consist of several components. Some of the components can be directly part of the product. This three-level structure is presented in Figure 8.

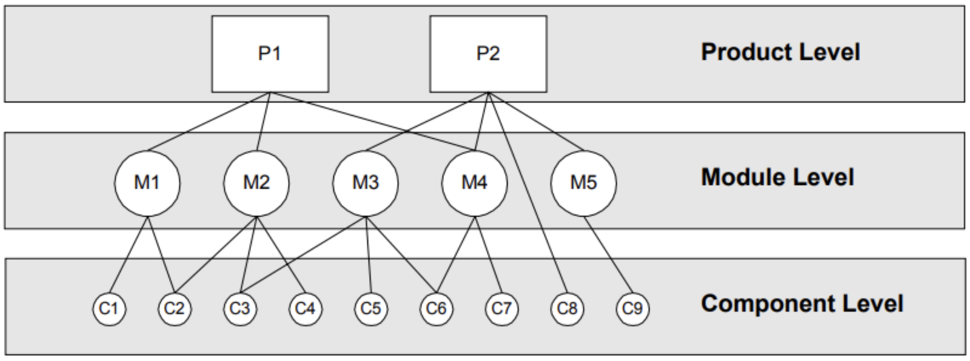


Figure 8 – Product structure model with three levels (Jørgensen K. A., 2001)

Functionalities encompass the functions or capabilities that a product offers to its users. Figure 9 shows the process of configuration where a customer chooses functionalities that define what modules and components will become part of the configured product and later physical model. Term *Attribute* is better describing component.

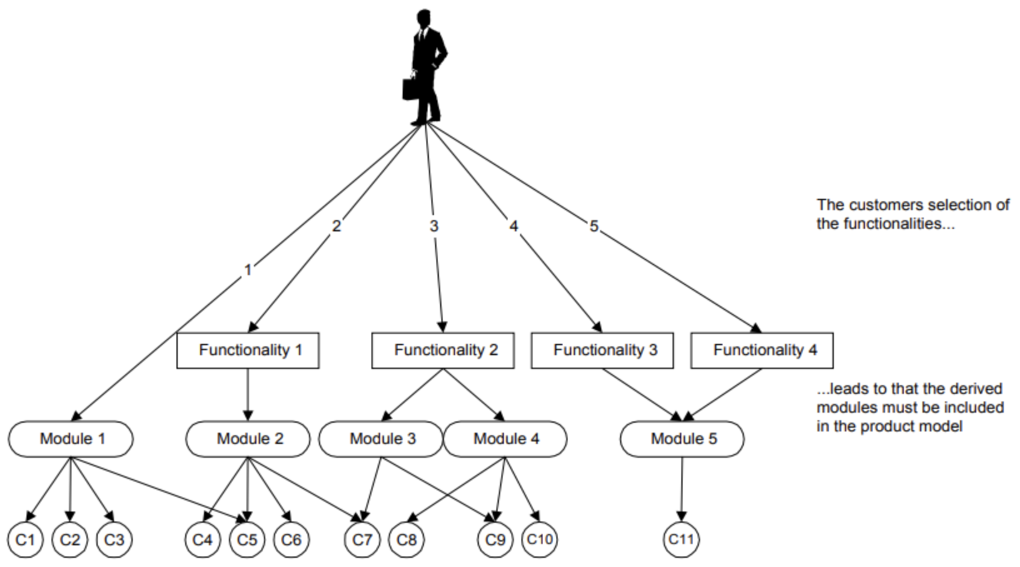
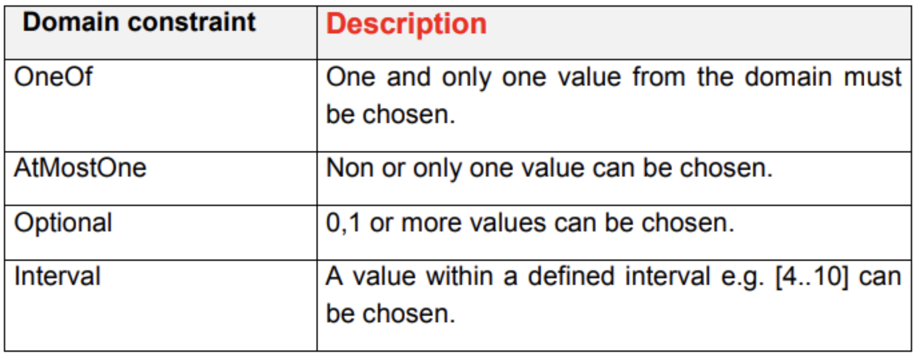


Figure 9 – Specification of modules through functionalities

Attributes are specified with *data type* (integer, real, boolean, text), *domain* defines the range of values that the property can have. Additionally, *domain constraints* represent logical rule on how the values in the domain shall be chosen, (Jørgensen, 2001), as presented in Figure 10.



Domain constraint	Description
OneOf	One and only one value from the domain must be chosen.
AtMostOne	Non or only one value can be chosen.
Optional	0,1 or more values can be chosen.
Interval	A value within a defined interval e.g. [4..10] can be chosen.

Figure 10 – Domain constraints for attributes that describe if and how the values in the domain shall be chosen. (Jørgensen K. A., 2001)

For a systematic approach and clear structure of data, syntax on how attributes are defined is adopted and shown in the following format :

Module type name (Attribute name, data type, domain constraint [domain]…)

For example, the implementation of this attribute definition in the case study will look as follows

Column (ColumnSection, String, OneOf [60x60, 80x80])

2.2.4. KBE - Knowledge-based engineering

Sriram recognizes different categories of design activity based on a process that leads to the desired outcome:

1. *Creative design,* where the prior plan for the problem solution does not exist but is approached throughout of process of creating multiple unique ideas, it is characterized by spontaneous, free-flowing thinking;
2. *Innovative design,* which combines existing elements or knowledge to develop a new product;
3. *Redesign,* where existing design is modified to meet changed functional needs and improve the solution's performance;
4. *Routine design*, where there is a prior plan for the solution, and it is based on existing solutions, (Sriram *et al.*, 1989). In this section, it is intended to look further into *Routine design*.

The term knowledge-based engineering (KBE), (Sandberg *et al.*, 2016), has become important in the manufacturing industry when dealing with automating routine design and repetitive tasks. KBE involves capturing, formalizing, and implementing engineers' knowledge into computer–based design applications. These applications can include fully automated processes as well as semi-automated processes that still expect some user interaction. KBE supports the design process by re-using predefined methods, algorithms, or results and by creating specific tasks and workflows. KBE is applicable in cases where the design is repetitive and can be expressed in explicit rules and like that formulated as a configuration problem, (Gembarski, Li, and Lachmayer, 2017) Configuration problem is characterized by the existence of components that have functional and technical properties with well-defined relationships between them and user requirements about functional characteristics. (Wurzinger, 2016)

In literature, levels of KBE with different levels of complexity are identified. It can go from the parametrization of geometric objects to complex software applications. In Figure 11, levels of complexity are distinguished, according to Hirz (Hirz *et al.*, 2013)

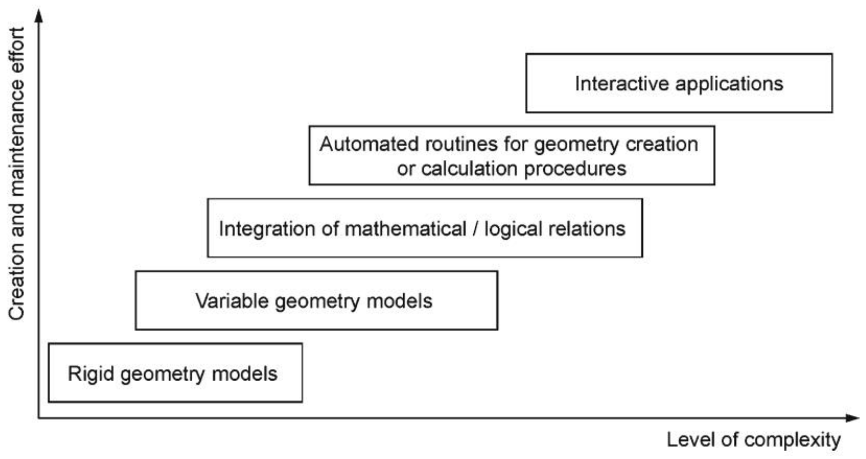

Creation and maintenance effort
Interactive applications
Automated routines for geometry creation or calculation procedures
Integration of mathematical / logical relations
Variable geometry models
Rigid geometry models
Level of complexity

Figure 11 – Different types of knowledge-based design applications (Hirz *et al.*, 2013)

The first category is "rigid geometry models", representing non-parametric 2D or 3D geometries found in databases. These models are simple, standardized components collected from previous projects, requiring minimal effort for creation and maintenance.

The next category is "Variable geometry models" or "Template models", which incorporate predefined geometrical driving parameters to allow for variability in geometrical characteristics. These models include parametric control of geometry-defining parameters, which users can edit. The creation and maintenance of variable geometry models is more complex due to the need to accommodate desired variation ranges and ensure stability.

"Integrating mathematical or logical relations" into variable geometry models enhances geometry creation by employing formulas, rules, reactions, or check operations. This integration enables control over geometry through input parameters and automated calculation of various parameters, such as material properties or weight. Automated routines are implemented using scripts, such as Visual Basic for Applications (VBA), adding complexity and requiring maintenance and updates.

Furthermore, "Interactive applications" are problem-oriented software solutions integrated within design software, featuring graphical user interfaces (GUI) for user-friendly interactions, parameter management, and integrated calculation or simulation procedures.

Integrating knowledge-based methods and tools into design processes offers significant support, improves development efficiency, and enhances product and process-related know-how.

For better and more structured implementation of KBE, the process must be supported by the methodology. MOKA is a methodology for the development of KBE application that consists of 6 steps, shown in Figure 12 (Stokes, 2001)

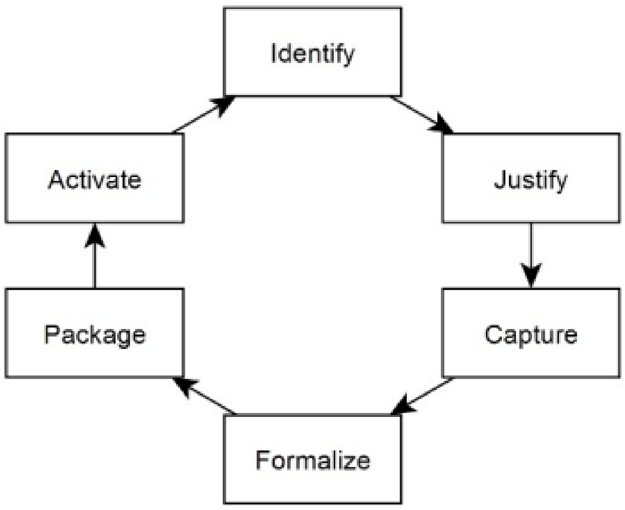


Figure 12 – The MOKA phases, adapted from (Stokes, 2001)

Identification: This phase focuses on defining the objectives, scope, and technical specifications for the application, at a concept level;

Justification: During this stage, a thorough examination is conducted to assess and analyze the commercial, cultural, and technical risks associated with the project;

Capture: During this phase, the primary focus is placed on the collection and systematic organization of fundamental engineering knowledge essential for the application, which is then modeled into an Informal Model. This model is structured of different components, including textual descriptions, and simple graphical representations like diagrams and sketches, which serve to closely explain concepts and relationships. Additionally, it includes Use Cases and Scenarios, along with Interview Transcripts, both of which describe knowledge, insights, and the decision-making process.

Formalization: In this phase, the Informal Model is transformed into a more structured Formal Model. The formal model is suitable for computer processing and automated reasoning. It is represented through sets of rules or logical statements that define relationships, constraints, and conditions within the knowledge domain. Formal model can include equations, formulas, algorithms, graph-based representations, etc.

Packaging: This phase involves the translation of the Formal Model into code, specifically for a Knowledge-Based Engineering (KBE) application.

Activation: The final stage encompasses the distribution, installation, and utilization of the developed application.

The Capture and Formalization phases are the most comprehensive steps in the process, and they are facilitated by using diagrams like Unified Modelling Language-based approach IDF0, etc. UML is a visual modeling language that has the purpose to illustrate how a system is designed its behavior and structure. The IDEF0 Functional Modelling approach has the purpose of depicting the choices, actions, and operations within an organization or system.

MOKA Methodology is recognized as suitable for this case study due to its well-structured phases and systematic approach to develop KBE application. This methodology helps with effectively structuring capturing and utilizing knowledge. The following steps were taken in line with MOKA framework to achieve the research objectives outlined in the Introduction. (Figure 13)

1. In the Identification phase company needs will be presented and analyzed, The Justify phase is not of importance for this case study due to lack of influence on the project itself.
2. In the Capture phase gathered knowledge will be presented and formalized in more structured diagrams. Gathered knowledge can be grouped into two categories first one is Process knowledge that supports the Identification phase and Knowledge about the structure, components, and functionalities of a greenhouse. Firstly, the design process will be captured and presented with an IDF0 diagram, followed by the suggested TO BE process. Next, diagrams and tables for modules and components will be established for better systematization and organization for later implementation in the code itself.

3. Chapter 4 will resemble the Packaging phase where the application will be presented and explained.

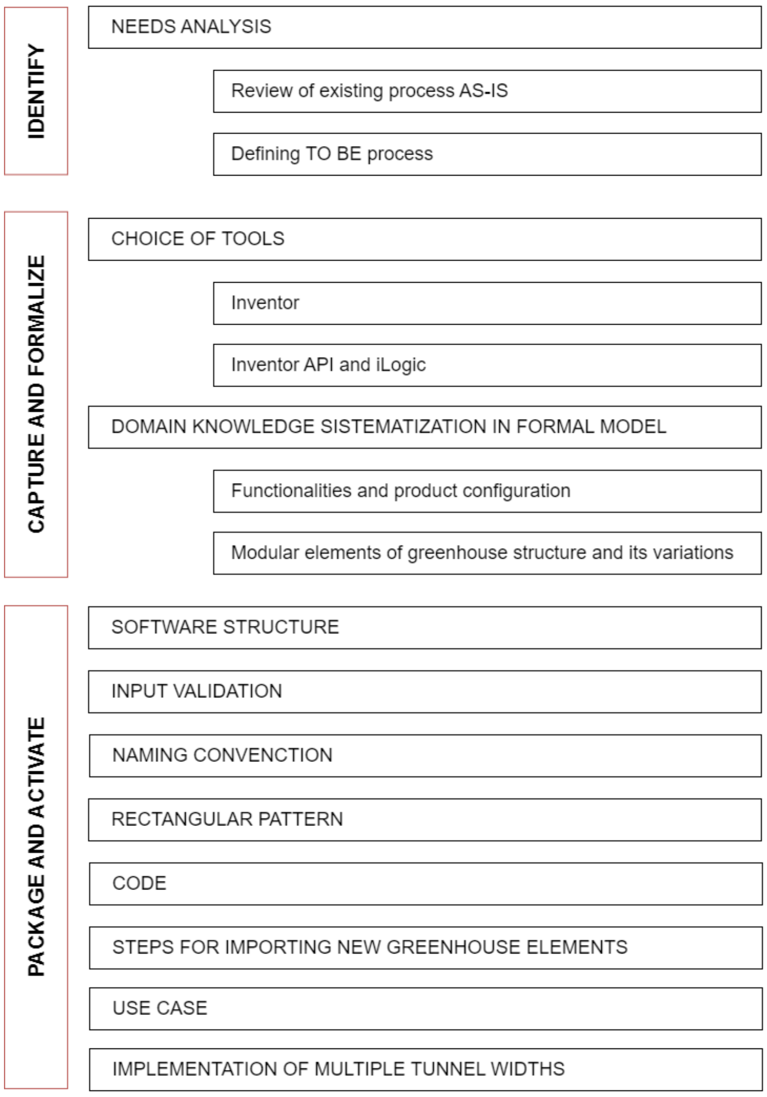

IDENTIFY
NEEDS ANALYSIS
Review of existing process AS-IS
Defining TO BE process
CAPTURE AND FORMALIZE
CHOICE OF TOOLS
Inventor
Inventor API and iLogic
DOMAIN KNOWLEDGE SISTEMATIZATION IN FORMAL MODEL
Functionalities and product configuration
Modular elements of greenhouse structure and its variations
PACKAGE AND ACTIVATE
SOFTWARE STRUCTURE
INPUT VALIDATION
NAMING CONVENCTION
RECTANGULAR PATTERN
CODE
STEPS FOR IMPORTING NEW GREENHOUSE ELEMENTS
USE CASE
IMPLEMENTATION OF MULTIPLE TUNNEL WIDTHS

Figure 13 – Case study through MOKA methodology

2.2.5. Parametric design

The construction industry has witnessed significant transformations due to advancements in technology. With the development of hardware, computational power became a powerful support in addressing increasingly complex demands faced by engineers. Furthermore, it serves as a tool in the automation of the design process that provides greater operational efficiency.

Computer Aided Engineering (CAE) has emerged as a crucial factor in providing reduced design time, producing prototypes faster, and achieving higher product quality (Randy Shih, 2014) Throughout history, several milestones have played important roles in shaping Computer–Aided Design (CAD) technology. The foundation was set by Sutherland in 1963 with 3D wireframe methods. Subsequently, advancements in surface modeling and solid modeling took place in the late 1970s. Two dominant methods for representing solid models are constructive solid geometry (CSG) representation and boundary representation (B-rep). In the late 1980s, *a feature–based parametric solid modeling* approach was introduced. (Monedero, 2000)

Parametric modeling is a widely used term in recent years, throughout the literature. It is suggested as one of the solutions that can provide greater design efficiency, faster iterations of the model variants, and better responsiveness towards the client. Oxam defines parametric modeling as the activity where the designer writes the code for the *parametric schema* in order to design the object. "Parametric schema" is defined as a unique mathematical model that supports the algorithmic process of shape generation (Oxman, 2017). Woodbry puts the accent on establishing relations between parts of the design, so that changes are coordinated, (Woodbury, 2010). In other words, parametric design is a computational approach to creating and modifying design objects by using parameters and rules to define relationships between elements. Parameters(variables) can define different aspects of a design such as size, shapes, position, etc. These parameters are interconnected through rules or equations that dictate how changes to one parameter affect others. Figure 14 shows how parametric design works. It begins with input data like shapes, points, and surfaces, combined with set parameters that can be different types, like Boolean, list, etc. These inputs go through a process where they're transformed based on parameter changes. The result of this process is a new design geometry that's created by adjusting those parameters. This approach, as explained earlier, lets designers easily change the design by modifying the parameters, which then influences how the final design looks and functions.

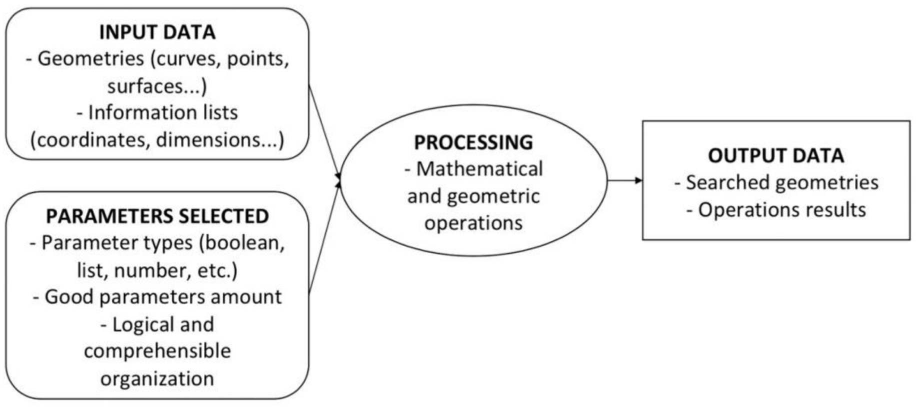

INPUT DATA
- Geometries (curves, points, surfaces...)
- Information lists (coordinates, dimensions...)
PARAMETERS SELECTED
- Parameter types (boolean, list, number, etc.)
- Good parameters amount
- Logical and comprehensible organization
PROCESSING
- Mathematical and geometric operations
OUTPUT DATA
- Searched geometries
- Operations results

Figure 14 – Operating principle of a parametric file (Girardet and Boton, 2021)

Automatic change propagation and geometry re-use are among the major benefits of using parametric design in opposite to rigid geometry (Shah, 2001) .

This page is intentionally left blank

3. METHODOLOGY

The first part of the chapter will elaborate upon the need for the development of a tool that automates the design phase. This will be followed by an examination of the existing process, commonly referred to as “AS IS” state. Subsequently, recommendations will be presented for the “TO BE” process.

In the second part knowledge collected through interviews, drawing analysis, and literature review will be organized and systematized for implementation in code during the later phase. Firstly tools that are being used for code development are being explained. Secondly, information about greenhouse structures is systematically collected and organized based on established principles derived from the manufacturing industry. These principles revolve around the comprehensive depiction of a product through its distinct attributes, modules, and components. This encompassing approach includes the creation of a product hierarchy, establishing the interrelationships between elements, and explaining the functional influences guiding the selection of particular modules and components.

3.1. NEEDS ANALYSIS

Estufasminho, a Portuguese company, has been devoted to manufacturing and installing agricultural greenhouses for numerous years. While possessing substantial experience in the industry, the company recognizes the strategic significance of incorporating new methodologies into their design and fabrication processes as they aspires to expand into the European marcet. The subject of this thesis will be the design phase, which is currently done manually. Therefore, the present work intends to implement parametric design and introduce new methodologies and workflows into the design process of greenhouses. With clearly defined modules and components structured as parametric library, workflows, and methodologies company could benefit from the automation of repetitive manual work, reduction of mistakes, faster responsivnes towards the client, and higher amount of projects done. Values of developing automated assembly generation files are time efficiency, achieved by reduction of manual effort, precision, and consistency ensured through a consistent approach to assembly generation and elimination of human mistakes in the design process.

This tool will generate 3D model for standardized inputs, and limit manual work to specific customization requests.

3.1.1. Review of existing process AS-IS

Figure 15a shows the typical project methodology employed in the company. It consists of the following phases:

Client Requirements Gathering Phase: The design process starts with acquiring comprehensive requirements from the client. This includes specifications, dimensions, materials, plot particulars, desired features, and any distinct prerequisites.

Preliminary Design Conceptualization: Subsequently, the design phase is initiated with the conceptualization of preliminary ideas. This involves the creation of hand-drawn sketches or digital concepts, aimed at facilitating the visualization of potential design options.

Foundation and Understanding Establishment: Following this, revisions are incorporated to ultimately establish a solid foundation and mutual comprehension with the client. This step paves the way for the phase of the design process.

Creation of Assembly File and Engineering Analysis: The development of an assembly file starts, wherein modeled components are meticulously assembled to form a comprehensive greenhouse structure within the Inventor assembly environment. Concurrently, engineering analyses are conducted to ascertain structural integrity, stability, and resilience against environmental loads.

Client Revision and Refinement: In this phase, the client is accorded the opportunity for revision. Leveraging the revision document and feedback from the client, the design process progresses toward the crafting of detailed design elements. Adjustments and customizations are implemented based on the client's input.

Detailed Design Presentation and Approval: Upon the creation of the detailed design, a presentation is orchestrated for client review and final approval. This presentation accommodates any last-minute alterations or adjustments as per the client's preferences. The finalized detailed design serves as the foundation for the subsequent steps.

Comprehensive Bill of Materials (BOM) Generation: A comprehensive Bill of Materials (BOM) is generated, defining all components, materials, quantities, and part numbers for the manufacturing process. The BOM serves as a pivotal instrument for material selection, budgetary considerations, and procurement of selected materials and components.

Technical Drawing Creation: Technical drawings are generated, following orthographic views, sectional perspectives, and exploded views. Step-by-step assembly instructions and installation guidelines, thereby offering invaluable guidance for both the manufacturing and assembly processes. Moreover, essential vector files tailored for the manufacturing process are produced.

Manufacturing Coordination and Field Assistance: Throughout the manufacturing process, seamless coordination is maintained, and the field team is supported during the construction phase.

Documentation Archival: As a final step, documentation archival is undertaken, ensuring the systematic preservation of all design files, documents, and drawings. This repository serves as an invaluable resource for future reference and potential enhancements to the design. Lessons learned are important steps that refer to insights, knowledge, and experiences gained from the project. By capturing and implementing them every following project will be improved.

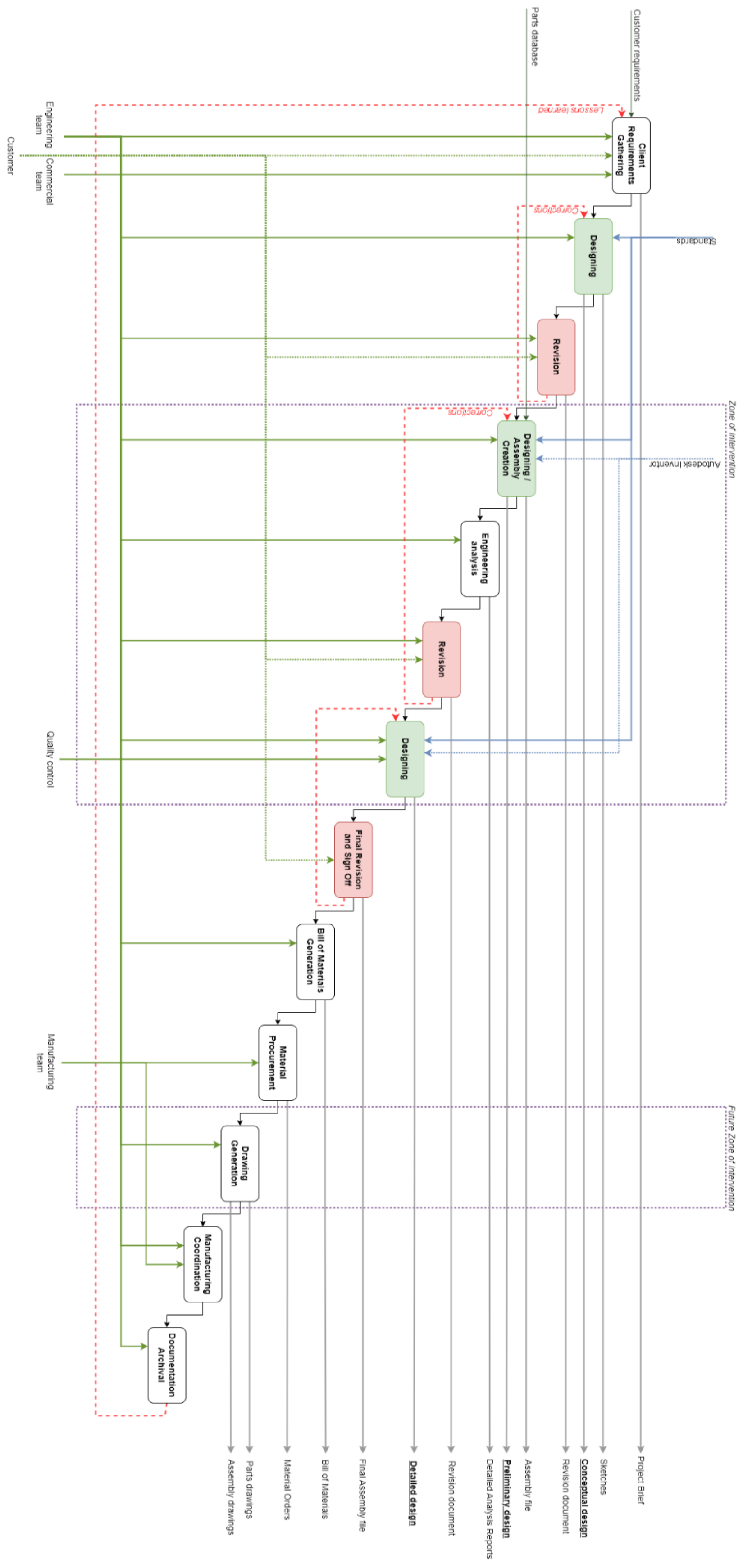


Figure 15a Diagram of existing process "AS-IS"

Figure 15 shows a diagram that illustrates Inputs, Mechanisms for process facilitation, and Controls including different standards that influence the design phase. Additionally, the diagram shows the resulting outputs of the process.

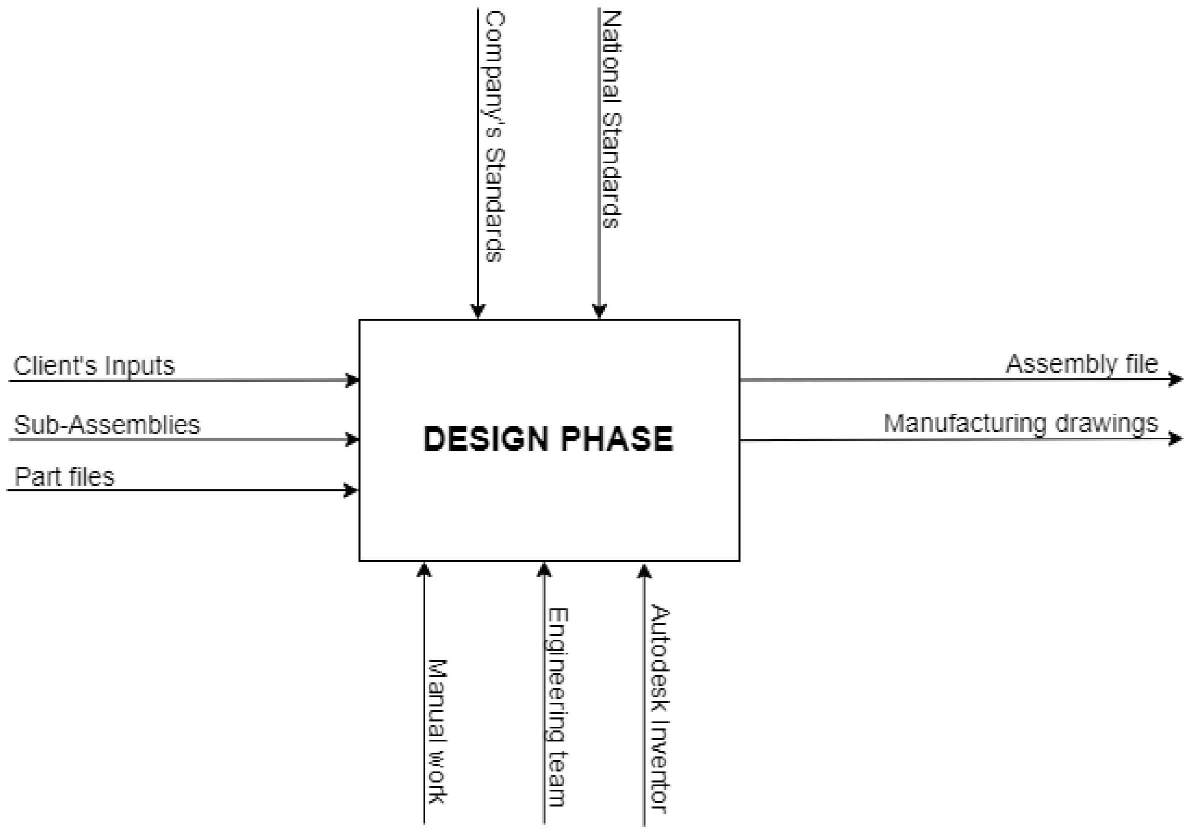


Figure 15 – "AS-IS" Diagram

3.1.2. Defining TO BE process

The main value in the defined TO BE process is the reduction of manual work by utilizing iLogic in Autodesk Inventor and automating the design phase. The suggested implementation is the use of the tool, that will facilitate automated assembly generation (see Figure 16).

Creating a parametric model consisting of essential structural elements of the greenhouse enables the quick generation of design variations and reduces the time and effort required to manually recreate designs from scratch. The possibility to fast simulate different options gives opportunities for better analysis of different design variations and for easier corrections of design. The main difference between the AS IS and the TO BE processes is the reduction of manual work leading to less time used for model generation. A parametric library of reusable components and modules is used as input, and Inventor API and iLogic External rule that make automation possible are introduced as mechanisms.

After the iterative process, code should be optimized and improved based on experience and lessons learned. Values achieved in this suggested TO BE phase are faster model generation, reduction of manual work, and the capability to create more iterations in less time.

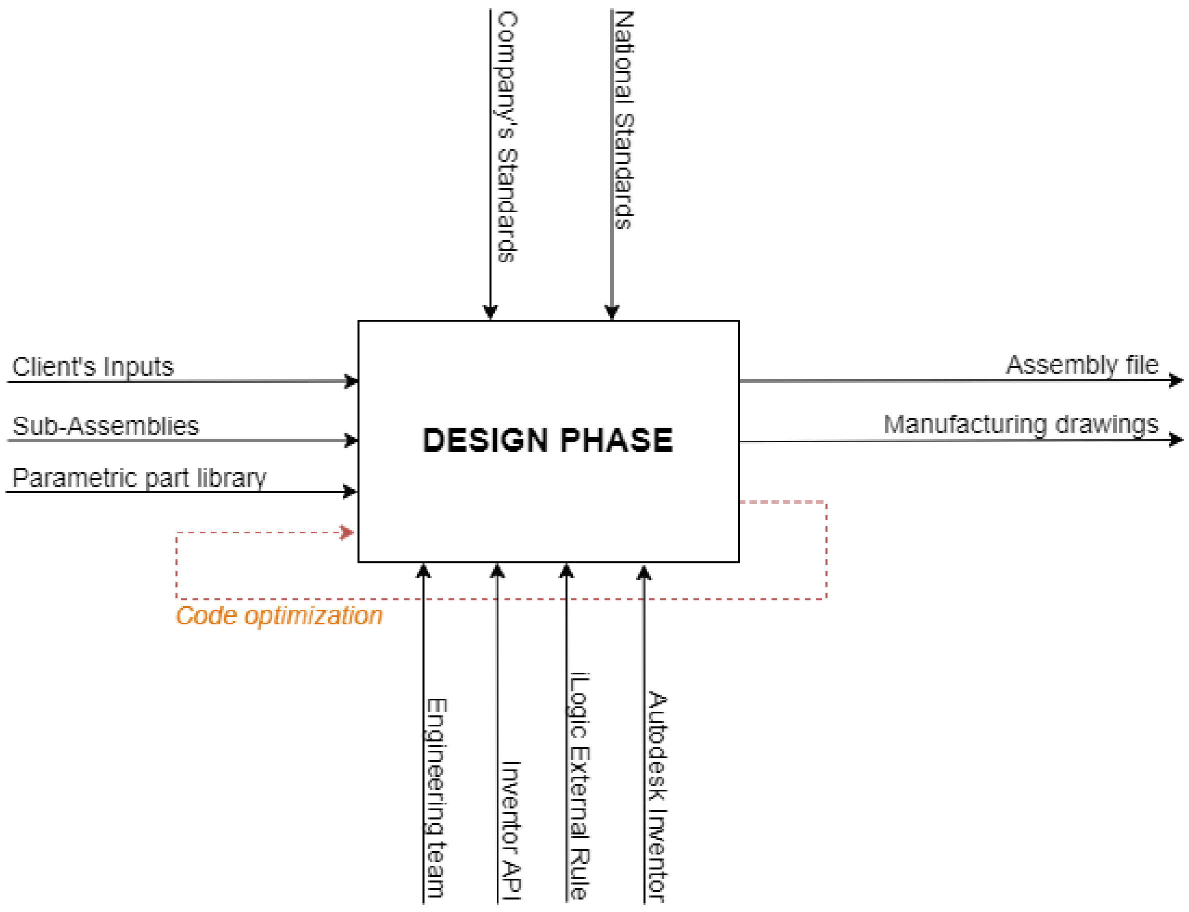


Figure 16 – "TO-BE" Diagram

3.2. CHOICE OF TOOLS

The selection of the program used for developing the case study was determined by the company and aligned with their established practices. Autodesk Inventor is the program for which the company holds a valid license and is currently being utilized for model development. The use of Inventor was adopted due to already pre-modeled parts and sub-assemblies that are being utilized in the greenhouses design. Up to now, Inventor has not been supported by any visual programming plugin like Dynamo, which imposes the use of the programming language VB.NET to access Inventor API and to achieve an automatization process.

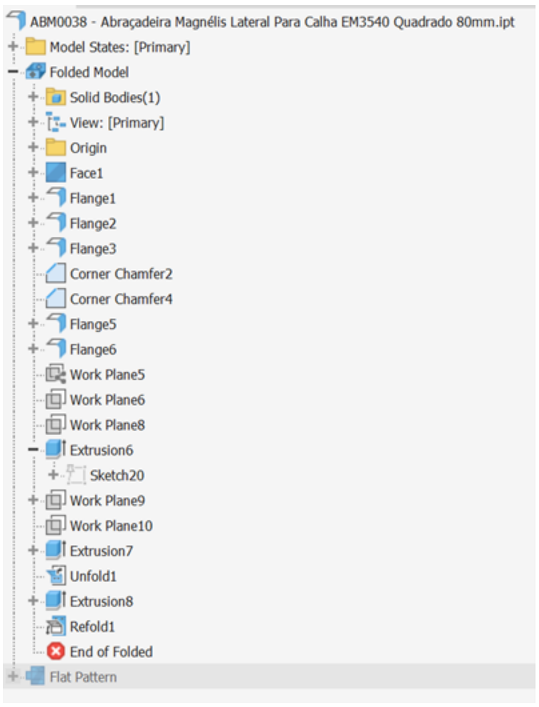
3.2.1. Inventor

Autodesk Inventor provides tools for the creation of a 3D mechanical design, the generation of manufacturing documentation, and the simulation of product behavior. It also includes features for stress analysis, frame design, and sheet metal design. It is a program that is detail-oriented and currently implemented in many industries among which are automotive, aerospace, construction, etc.

Inventor is classified as a history-based and feature–based parametric modelling program (Janssen and Stouffs, 2015). The history-based approach that is also referred to as the procedural approach is based on a sequence of operations used to generate a model. Those operations are visible in the Model browser (Figure 17). The model browser provides a clear overview on how the model is constructed, and it allows users to:

- Edit features or sketches at any point in the history and to modify the design,
- Suppress or unsuppress features to see the impact of their inclusion on the design, and
- Edit Parameters to adjust dimensions and other variables in a controlled manner.

This history-based approach enhances design flexibility and iteration, making it easier to adapt to changes and explore design alternatives.



```
ABM0038 - Abraçadeira Magnélis Lateral Para Calha EM3540 Quadrado 80mm.ipt
+ Model States: [Primary]
- Folded Model
    + Solid Bodies(1)
    + View: [Primary]
    + Origin
    + Face1
    + Flange1
    + Flange2
    + Flange3
      Corner Chamfer2
      Corner Chamfer4
    + Flange5
    + Flange6
      Work Plane5
      Work Plane6
      Work Plane8
    - Extrusion6
        + Sketch20
    + Work Plane9
      Work Plane10
    + Extrusion7
      Unfold1
    + Extrusion8
      Refold1
      End of Folded
+ Flat Pattern
```

Figure 17 – Model browser

Inventor is classified as feature-based parametric modelling software(Janssen and Stouffs, 2015). The feature is a predefined part or construction tool for which the user defines the key parameter, (Randy Shih, 2014). Examples of features are extrusion, holes, fillets and chamfers, patterns, etc.

The main building blocks of Inventor are parts (.ipt), which are modelled from 2D sketches. In Autodesk Inventor, sketching is a fundamental step in creating 3D models. Sketches are 2D profiles that define the shape and geometry of features in a part. These profiles can include lines, arcs, circles, rectangles, and more. Sketch Constraints maintain the relationships between sketch elements. Constraints ensure that certain geometric properties are upheld as the sketch is being edited and manipulated. Some common sketch constraints are Parallel, Perpendicular, Tangent, etc.

Inventor's assembly environment (.iam) allows the creation of complex structures by assembling multiple parts. When the part is imported into the assembly file, the link is created with the part location, meaning that the name or part location should not be changed, otherwise, an error occurs and the part needs to be located manually. This process involves applying various constraints to define the interactions between parts. Some common assembly constraints include "Mate", which aligns faces or edges together while maintaining their relative position, "Flush", which ensures that two surfaces or axes share the same center point, "Insert", which places one part into another, such as inserting a pin into a hole, "Angle", which defines an angular relationship between two parts.

An important part of every file is the Origin folder. Origin folder consists of origin planes, origin axes and single origin point. (Waguespack Curtis, 2013). It establishes a coordinate system and represents a stable foundation to anchor design, whether the sketch or the part, within the assembly file. A good practice is to build parts around the origin geometry whenever it is possible, and later when importing it into assembly to constrain the origin geometry of parts together. In that way, changes in part features will not affect constraints.

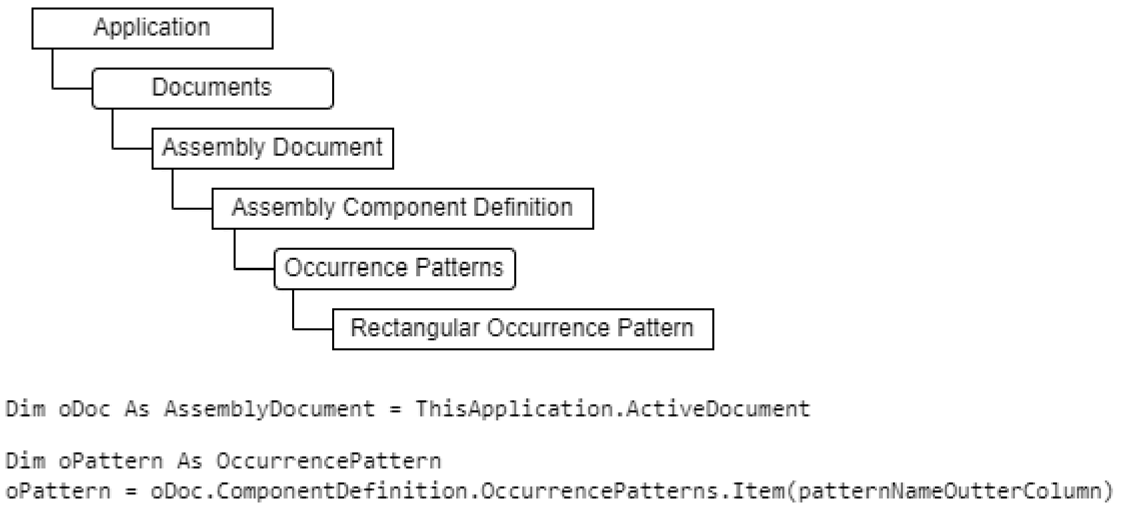
3.2.2. Inventor API and iLogic

The API (Application Programming Interface) presents a set of mechanisms, such as objects, functions, links, etc, that allow communication between two software components, providing an addition of functionalities that are specific to the users' needs (AWS, no date). Inventor API and Ilogic are two essential elements for automation of the design process in Autodesk Inventor.

The Inventor API is a set of pre-built code components, resources, functions, and documentation provided by Autodesk for users to interact programmatically with Autodesk Inventor.

Inventor API exposes the application's functionality in an object-oriented manner that represents the components and features of Autodesk Inventor's design environment(Autodesk, no date). These objects correspond to elements such as parts, assemblies, sketches, features, and more. Developers use these objects to manipulate and automate tasks within Inventor. Inventor API is organized through *object model*. Object model is a hierarchical diagram that illustrates relationships between objects. Objects are supported by properties (attributes) that closely define them and methods (e.g., move, delete, etc.) that represent instructions that the object understands. Various methods and properties of objects can be used through API.

The object that should be utilized in code is accessed by navigating through the structure hierarchically from the top object to one that is specifically needed. The application object represents the Inventor application and is the top-most object in the hierarchy. Figure 18 shows how a segment of the object model helps to call a particular pattern from the Assembly file.



```
Dim oDoc As AssemblyDocument = ThisApplication.ActiveDocument

Dim oPattern As OccurrencePattern
oPattern = oDoc.ComponentDefinition.OccurrencePatterns.Item(patternNameOutterColumn)
```

Figure 18 – Traversing the Object Model

`Dim oDoc As AssemblyDocument = ThisApplication.ActiveDocument`	(1)

Line (1) declares a variable named “oDoc” of type “AssemblyDocument” and assigns it the value of the currently active document within the Inventor application. “ThisApplication” refers to the Inventor application, and “ActiveDocument” is a property that returns the currently open document, which is an assembly document in this case, allowing to bypass the Documents object.

`Dim oPattern As OccurrencePattern`	(2)

Line (2) declares a variable named “oPattern” of type “OccurrencePattern”. An “OccurrencePattern” represents a pattern of component occurrences (instances) within an assembly.

`oPattern = oDoc.ComponentDefinition.OccurrencePatterns.Item (patternNameOutterColumn)`	(3)

Line (3) assigns a value to the “oPattern” variable. It does this by accessing the “ComponentDefinition” property of the “oDoc” assembly document. The “ComponentDefinition” property refers to the definition of the components within the assembly. Within the component definition, it accesses an “OccurrencePattern” using the *Item method*. Occurrence Patterns is a collection objects, meaning that it provides access to a group of related objects, by supporting Count and Item properties. The Item property is used to retrieve a specific occurrence pattern based on its name. The parameter “patternNameOutterColumn” is a variable that holds the name of the occurrence pattern being targeted.

In short, this code takes the currently active assembly document, accesses a specific occurrence pattern within the assembly's component definition, based on its name (held in “patternNameOutterColumn”), and stores this occurrence pattern in the “oPattern” variable for further manipulation or analysis.

Access to the API can be achieved through Standalone EXE, Inventor (Add-In or VBA), and Apprentice server (Figure 19) (Autodesk, no date). The blue cylinder symbolizes the data being accessed, such as Inventor files. Add-ins, VBA, and Inventor operate within the same process. VBA is a programming environment that can be accessed within Inventor. Add-ins are specialized software components that enhance the functionality of an existing program. They are commonly used in software applications to provide additional features, tools, or capabilities that might not be present in the core program. Add-ins are designed to seamlessly integrate into the user interface of the host application, allowing them to extend the application's capabilities and offer a more tailored experience for users.

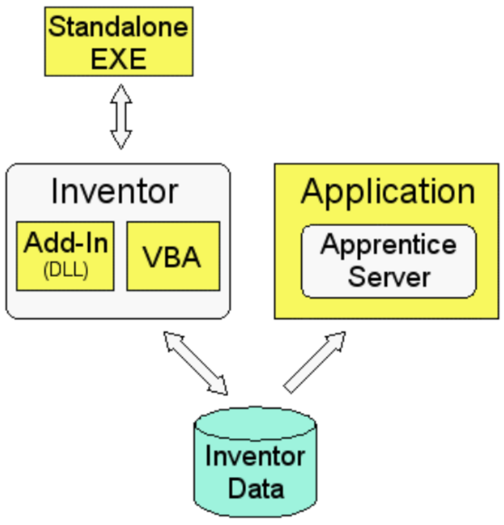


Figure 19 – Ways to access Inventor API (Source: https://help.autodesk.com/)

iLogic serves as an integrated add-in within Autodesk Inventor, facilitating the creation of customized rules using the VB.NET programming language. Rules represent a small Visual Basic program, these rules can be both internal, operating within the Inventor environment, and external, running outside of Inventor as standalone scripts. The tool offers a range of code snippets that help scripting by providing shortcuts for frequently used objects. However, iLogic has its constraints, a lack of debugging features typically found in programming environments might create challenges in identifying and solving errors within the created rules. Since iLogic is using VB.NET, code can be written in Visual Studio in a VB.net project. However, iLogic has already installed a library that provides shortcuts like "ThisApplication", "ActiveDocument" to work. In Visual Studio, these shortcuts should be adjusted for the program to work. This means that this process will only work from Visual Studio where code can be written debugged and copied to iLogic, for other direction, the code would need adjustments.

3.3. DOMAIN KNOWLEDGE SYSTEMATIZATION

3.3.1. Functionalities and product configuration

The configuration of a greenhouse structure depends on the identification of available functionalities that a customer can opt for. The chosen functionalities shape the design of a final product in the sense that each option requires the correct set of modules and components that support the chosen functionality.

Drawing parallels with Autodesk Inventor, the main assembly file presents the product level. The selection of functionalities defines sub-assemblies, each fulfilling a distinct functionality. In some instances, a given functionality will be satisfied by incorporating a single part, for example column. Figure 20 shows file structure of main assembly.

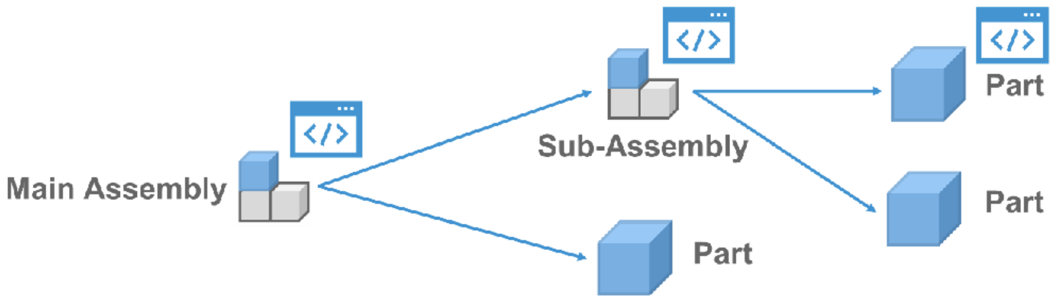


Figure 20 – Structure of a file (Ali Demir, 2021)

The Functionalities that are accessible to the greenhouse customers include:

1. **Type of tunnel roof:** The choice between "Rounded-arched" and "Gothic-arched" determines the foundational form of the greenhouse structure. Usually, all tunnels included in one greenhouse go with the same roof type.

2. **Number of tunnels:** The number of tunnels contributes to the overall layout of the greenhouse. It is customizable by the client and usually defined by the better usability of the plot. The last tunnel can differ from the others for better plot usage.

3. **Width of the tunnel:** Customers can select from a pre-defined range of tunnel widths, such as 8m, 9m, 10m, and 12m, influencing the overall dimensions of the greenhouse. Usually, a combination of multiple widths is used for better plot usage.

4. **Greenhouse length:** Predetermined by the plot, this parameter influences linear elements like gutters and horizontal profiles.

5. **Doors:** Customers have autonomy in defining door specifications, number of doors, measurements, and positions.

6. **Column section:** The columns size can vary between 60×60, 80×80, and 80×120.

7. **Column height:** Adjustable within predefined values (2500mm, 3000mm, 3500mm, 4000mm, 4500mm, and 5000mm), this parameter complements the column section in defining column properties. And is making it possible to influence different heights of greenhouse.

Figure 21 explains how the choice of certain functionalities influences the option for a particular module, or in other words, what functionalities need to be defined for the module to be determined. Product consist of modules, and modules consist of components. This three-level structure when applied to file structure looks as schemetically presented in Figure 20. Main assembly represents product level, sub-assemblies represent module level, and part level represents component level.

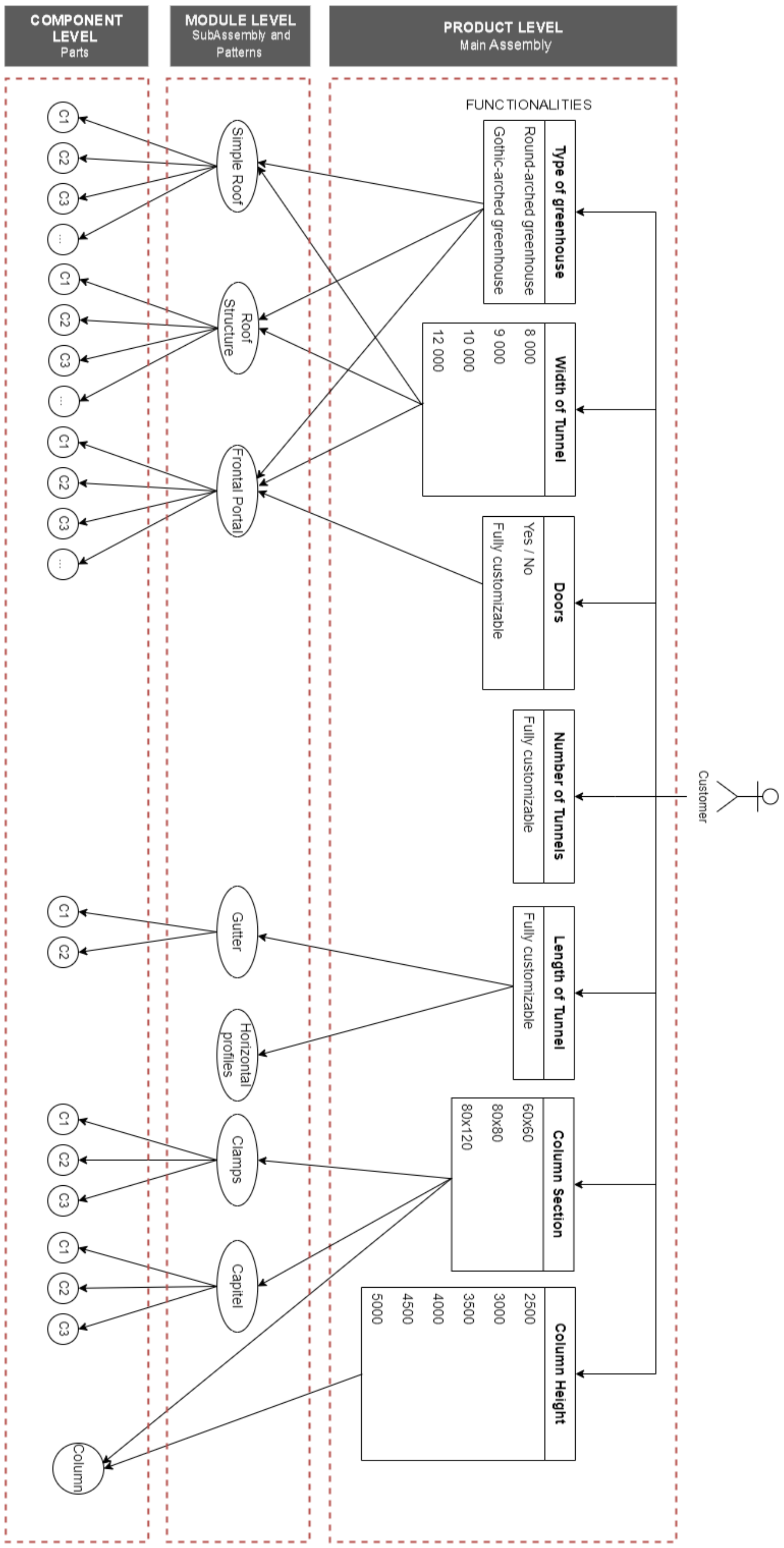


Figure 21 – Diagram of functionalities and influenced modules

3.3.2. Greenhouse elements

Elements included in this study are put down to eight exemplary elements. Figure 22, shows these elements in 3D view, while Figure 23 is showing plan view of how are element organized and what are the distances between them:

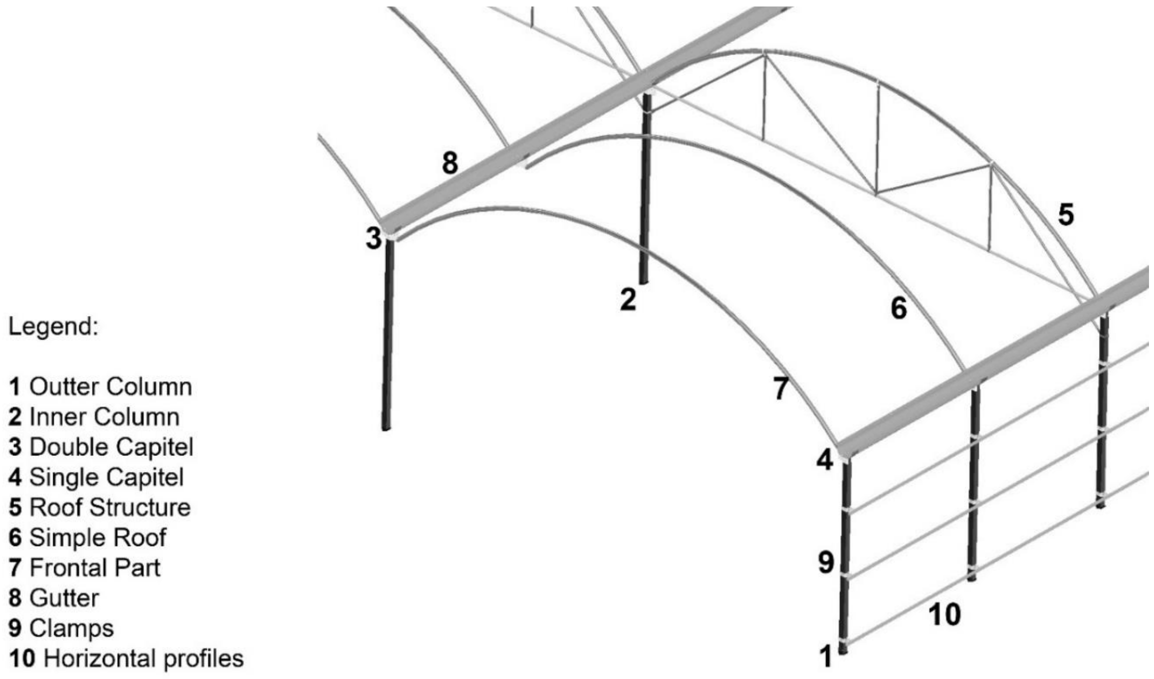


Figure 22 – Greenhouse elements included in the case study

1. **Outer Column:** The outer column is determined by the chosen *Column section* and *Height*. It is defined as a parametric part whose dimensions are controlled by the local rule (Rule inside the Column part file) that is being run from the Main Assembly file. The distance of outer columns in the Y direction is 2500mm. The distance of the columns from the 2nd line till n-1 in the Y direction is 5000mm.

2. **Inner Column:** Column that is positioned inside of the greenhouse. It has the same dimensions as the outer column, but its thickness is 2mm, unlike the outer column, which is 3mm thick.

3. **Double Capitel** is positioned in lines from 2nd till n-1. Its dimensions are defined by the *Column section.* Due to its complexity, these elements are being added to the Main Assembly as a sub-assembly. What sub-assembly will be added is determined based on chosen functionalities.

4. **Single Capitel:** This is an element that is positioned on the sides of the greenhouse. It is imported as a double capitel and influenced by the same functionalities as a double capitel. Distances in Y direction are 2500mm.

5. **Roof Structure:** These modules can be determined when both the *Type of greenhouse* and *Width of the tunnel* are determined. The roof structure contributes to the overall architectural design. Roof Structure starts from the 3rd outer column in the Y direction and goes with every odd column number.

6. **Simple Roof:** This module is determined in the same manner as the roof structure and is positioned after the frontal portal as every even roof, while in between is the roof structure.

7. **Frontal Portal:** Dictated by the *Width of the tunnel*, *Type of greenhouse*, and presence of a *Door*, the frontal portal shapes the entrance. It is aligned with the first and last columns.

8. **Gutter:** These elements are defined by the *Greenhouse length*

9. **Clamps:** Defined by the *Column section*, the number of clamps is determined by the *Column Height.*

10. **Horizontal profiles:** Defined by *Greenhouse length*, these elements are connected to clamps and their number is influenced by *Column Height.*

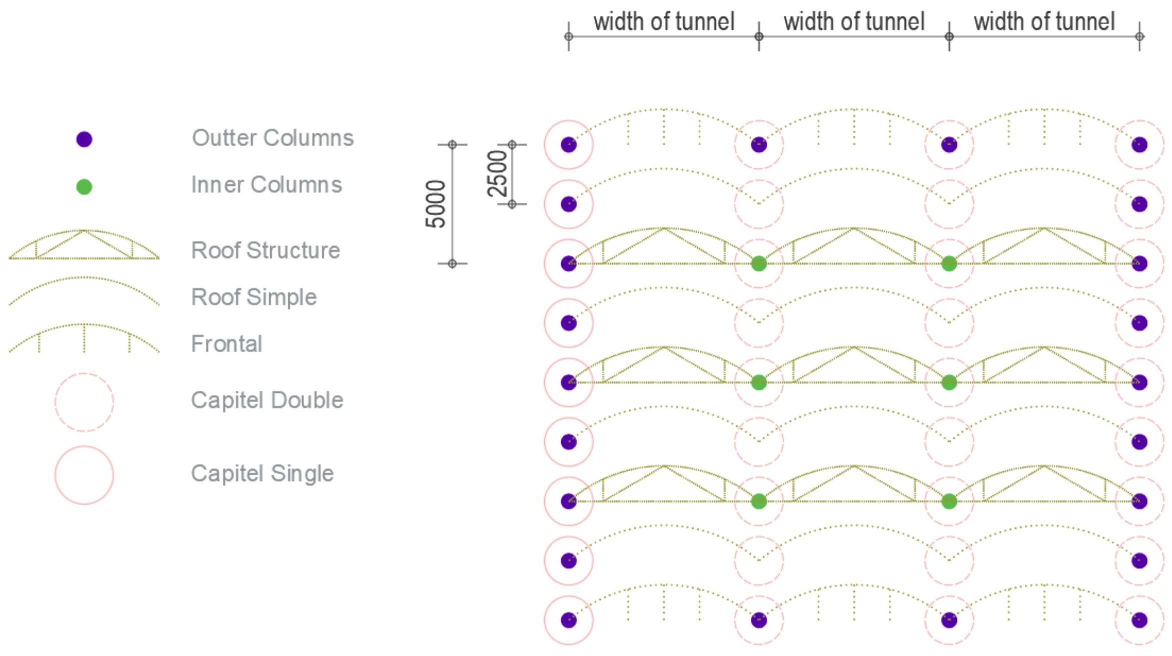


Figure 23 – Grid of elements in greenhouse

3.3.3. Modular elements of greenhouse structure and its variations

The greenhouse structure is made of modular elements. An important step is to organize, describe, and understand how module variants are achieved. Each variant module and component is solved with one of three approaches, depending on its geometry and complexity.

The first approach deals with a case of complex geometry that cannot be influenced by parameters. For example, a curvature of the roof arc is achieved through many polygons that cannot be easily controlled by parameters. In this case, the sub-assembly defined by choice of functionality is directly imported into the main assembly. Elements imported in this way into the main assembly are double capitel, single capitel, roof structure, simple roof, and frontal part.

The second approach is used in elements that directly satisfy functionality (Height and Column Section), the columns. Variations of these elements are regulated by triggering a rule inside the part file that changes the geometry of the element. These elements are simple enough in geometry and can be easily manipulated by updating parameters. The first snippet of code, lines (4), shows a method to run the rule inside the part file, with defined variables and its data type. The second snippet, line (5), shows how the code is implemented in the case study. “BaseNameColumnOutter” is a variable that holds the name of the column part while “ColumnDimensions” represents the name of the rule that is being triggered.

```
Function RunRule (
      componentOrDocName As Object,
      ruleName As String
) As Integer
```
(4)

```
iLogicVb.RunRule(baseNameColumnOutter, "ColumnDimensions")
```
(5)

The third approach is applied on a case of sub-assemblies that consist of simple parts whose dimensions can be manipulated through parameters. The element of this category is a clamp. Variation of this element is solved with model state. Model state is a functionality introduced inside of Inventor 2022 that allows the creation of different states of one part. For example, model state can help define the level of details, manufacturing state variations of one element, etc. Each model state can contain different dimensions, features, components, parameters, etc.

Some component variations are solved by combining two of the mentioned approaches, like in the case of columns. Within the column part file, column section and column height are defined with the rule, while column thickness is defined with model states. Later, in the main assembly, the right model state is imported to the right position within the structure. Using model state, instead of uploading two different components, is generally more memory-efficient and reduces the need to duplicate geometry. Importing separate parts for each configuration can lead to larger file sizes and increased storage requirements. Figure 24 presents Column model states, with thickness variation, the first model state is named “Outer” and is used for outer columns, while the second one is “Inner”, used for inner columns.

Elements like gutter and horizontal profile are imported as elements with fixed dimensions.

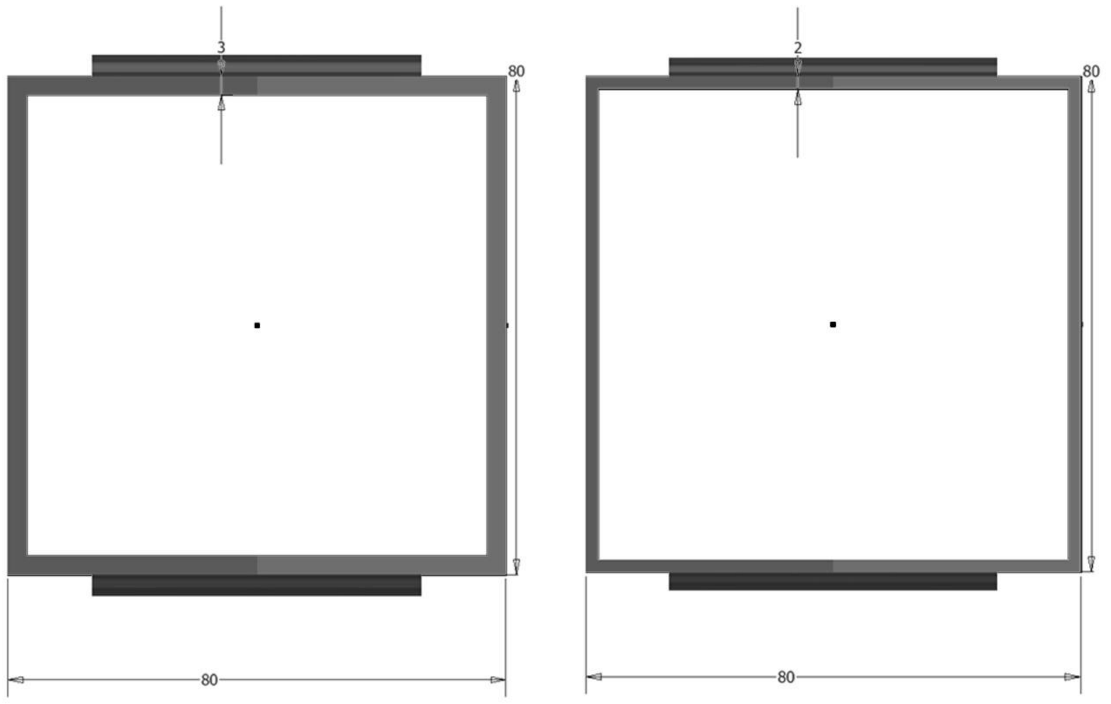


Figure 24 – Column model states - Outter and Inner

Line (6) is from Inventor API documentation. It explains the method to add components with specific model states. Line (7), shows code adjusted to the case study, where the variables used are component name, file location, and position where the column is placed in the main-assembly file.

```
Function AddWithModelState (
      occName As String,
      file As String,
      modelStateName As String,
      Optional position As PointOrMatrix = Nothing,
      Optional grounded As Nullable(Of Boolean) = Nothing,
      Optional visible As Nullable(Of Boolean) = Nothing,
      Optional appearance As StringOrAsset = Nothing
) As ManagedComponentOccurrence
```
(6)

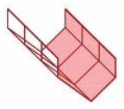
```
Components.AddWithModelState(
 "ColumnInner",
 PartsFolder & "\PG808024700 - Pilar Galvanizado Quadrado
80x2.0mm.ipt",
```
(7)

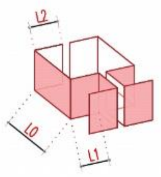
`"Inner",`
`pos)`	

Table 1 and Table 2 present elements of the greenhouse and their properties. For better systematization, each element has an ID card where all possible properties are written and clearly explained. Each element is described with the properties, that show what variations of certain parts or subassemblies are possible. To better define properties, the next categories are used: property name, data type of the property, values, and how that value is chosen within a certain domain. Functionality defines whether that property is possible to be directly chosen by the customer. “Influenced by” is information in the table that says what functionality is defining and influencing certain parts or sub-assemblies to be chosen. For better overview and control, the last information about parts or assemblies is the Name for import, meaning how and with what name that assembly and part is imported in script. Having all these elements in one place helps to have better control over the database of the elements used in the software.

Table 1 – Table of elements

Horizontal profile	PropertyName	Data type	Domain	Functionality
	Length	Integer	Fixed Value	No
	Name for import:			
	Influenced by:	*GreenhouseLength*		
Imported as part				

Gutter	PropertyName	Data type	Domain	Functionality
	Length	Integer	Fixed Value	No
	Name for import:			
	Influenced by:	*GreenhouseLength*		
Imported as part				

Clamps	PropertyName	Data type	Domain	Functionality
	ColumnSection	String	OneOf [60x60, 80x80]	No
	Name for import:			
	Influenced by:	*GreenhouseLength*		
Sub-Assembly / Model State				

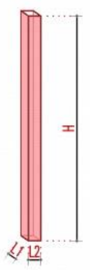
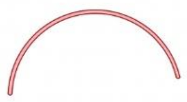
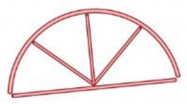
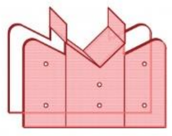
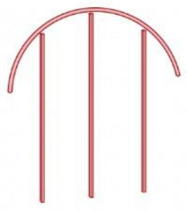
Column	PropertyName	Data type	Domain	Functionality
	ColumnSection	String	OneOf [60x60, 80x80]	Yes
	ColumnHeight	Integer	OneOf [2500,3000,3500,4000,4500,5000]	Yes
	ColumnThickness	Integer	OneOf [2, 3]	No
	Name for import:			
	Influenced by:	*ColumnSection, ColumnHeight*		
Local rule / Model state				

Table 2 – Table of elements

Simple roof	PropertyName	Data type	Domain	Functionality
	Width	String	OneOf [8000, 9000,10000,12000]	No
	Name for import:			
	Influenced by:	*TypeofGreenhouse, WidthOfTunnel*		
Sub - Assembly				

Roof structure	PropertyName	Data type	Domain	Functionality
	Width	String	OneOf [8000, 9000,10000,12000]	No
	Name for import:			
	Influenced by:	*TypeofGreenhouse, WidthOfTunnel*		
Sub - Assembly				

Capitel	PropertyName	Data type	Domain	Functionality
	Variation	String	OneOf [Double, Left, Right]	No
	Section	String	OneOf [60x60, 80x80]	No
	Name for import:			
	Influenced by:	*ColumnSection*		
Sub - Assembly				

Simple roof	PropertyName	Data type	Domain	Functionality
	Width	String	OneOf [8000, 9000,10000,12000]	No
	Name for import:			
	Influenced by:	*TypeofGreenhouse, WidthOfTunnel*		
Sub - Assembly				

This page is intentionally left blank

4. SOFTWARE DESIGN

4.1. SOFTWARE STRUCTURE

Figure 25 illustrates the software components and their relations within the system. The software system consists of four components: Inventor and Inventor API, iLogic, and the components folder. The design engineer opens Inventor, which in this case is used to start iLogic. iLogic is an Add-in that runs in the same process as Inventor and is used to access functionalities of Inventor through Inventor API. Through iLogic, the design engineer runs the rule that starts the process of assembly generation. The components folder, placed on the local machine, works as the parametric database. It contains all part files and sub-assemblies used for main-assembly generation. iLogic establishes communication with the components folder, retrieving necessary components and integrating them into the modelling environment. To ensure the successful execution of the code, it is imperative to specify the location of this components folder. Within the primary folder, distinct subfolders contain the specified sub-assemblies. Since exact folder names, parts file names, and sub-folders names are used for software to communicate with components folder, the naming of the folders and all the other elements should be consistent, as exemplified in Figure 26.

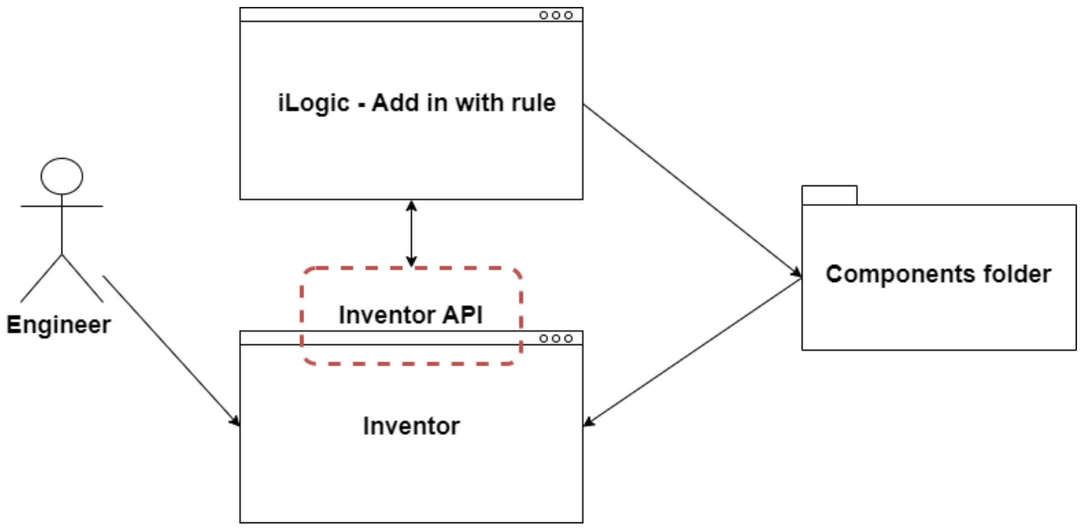


Figure 25 – Software system

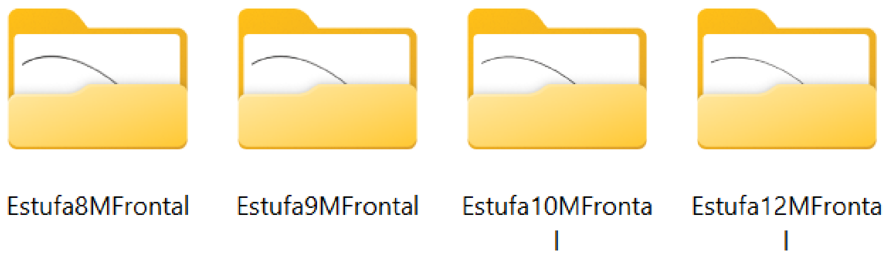


Figure 26 – Example of folder organization

To make code more reliable, the use of constraints in the code should be minimized. Positioning of the parts and sub-assemblies should be defined as precisely as possible inside the files of parts and sub-assemblies. If the part geometry is changed or faces that are being constrained are named differently of removed from the part or sub-assembly, the code will submit an error. Due to that reason, part files and sub-assemblies should be prepared and modelled in the right way. Coordinates of elements preferably should be made to be aligned with elements that are in relation with (see Figure 27). Columns, capitels, and clamps should be cantered with origin planes. XZ plane should be symmetrically constrained within the sub-assemblies of roof structures and frontal sub-assemblies. These recommendations are important, so the generated model results as precise as possible.

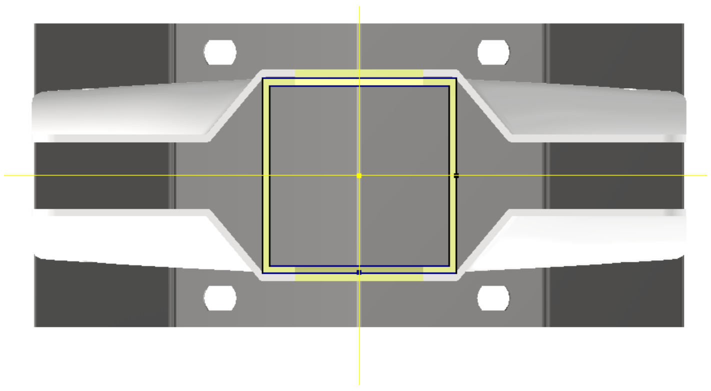


Figure 27 – Centering parts and Sub - Assemblies according to Origin planes

In Figure 28, software is analyzed with the aid of a UML sequence diagram. This diagram analyses communication between components over time and the sequence in which these interactions occur. Vertical rectangles of the diagram represent the time during which that certain component is active. The first step is input generation by the engineer throughout the input forms. Inputs are sent into variables in iLogic, after which iLogic communicates with the component folder to get the right parts and sub-assemblies for greenhouse generation. From the folder, components are placed in model space and pattern is made.

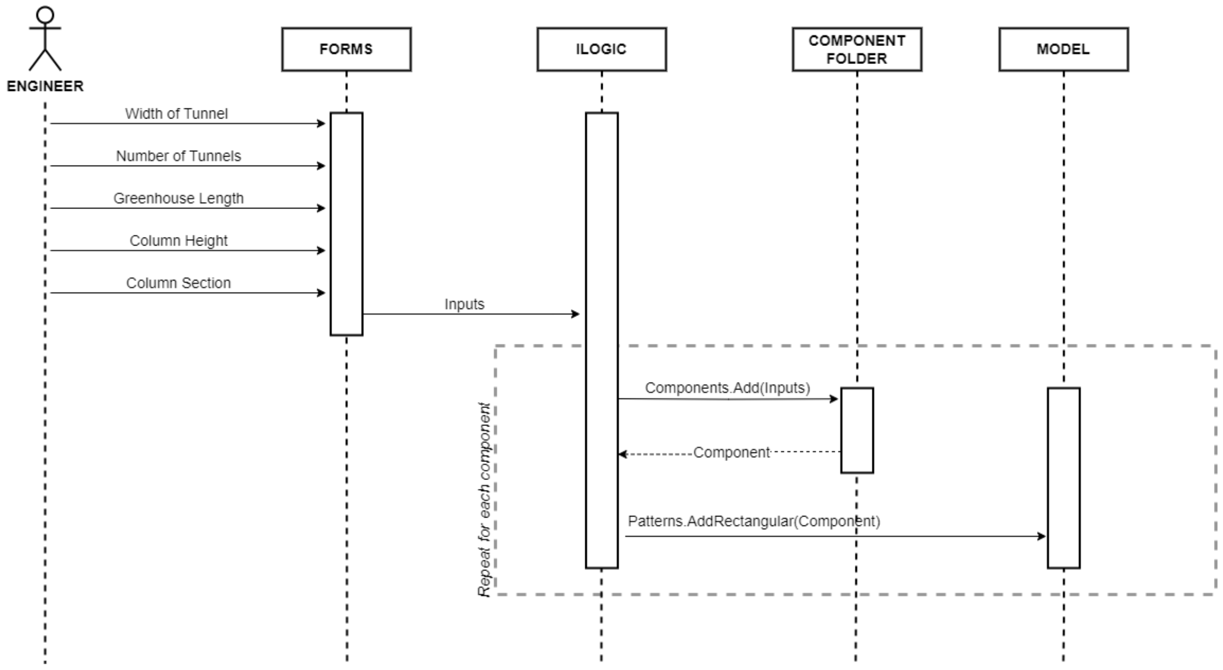


Figure 28 – Code Structure presented with UML diagram

4.2. INPUT VALIDATION

To optimize the program's effectiveness, inputs should be specified in the following manner:

- Component Folder – presents the input that establishes a connection with the code and content folder from where code will take all necessary parts and sub-assemblies to configure the greenhouse. It should be in the form of the link like in example *D:\01 Academia BIM\BIM A+7\Parametrizacija*
- Width of the tunnel – The user is choosing one value from a multi-value list with values 8000mm, 9000mm, 10000mm, and 12000mm presented as an input list box. Widths are presented as values in mm. The default option is 10000 mm. In case the user proceeds without making a selection, the program will automatically assign the default width of 10000 mm to the variable.
- Number of Tunnels – The user is required to input an integer value representing the desired number of tunnels. The minimum permissible value is 2.
- Greenhouse Length – Valid input is a number in the value of an integer, without space, presented in mm. In the current stage for optimal outcomes, the value should be divisible by 2500. However, if this rule is not met, manual adjustments are necessary.
- Column Height – The user is choosing one value from the multi-value list that includes 2500mm, 3000mm, 3500mm, 4000mm, 4500mm, and 5000mm. The default choice is 3000 mm.
- Column Section – The user has the option to choose between two column section sizes: 60×60 and 80×80. If no selection is made, the program will proceed with the default choice of 60×60.

4.3. NAMING CONVENTION

For easier navigation throughout the code, a consistent naming convention is important. The following convention is adopted:

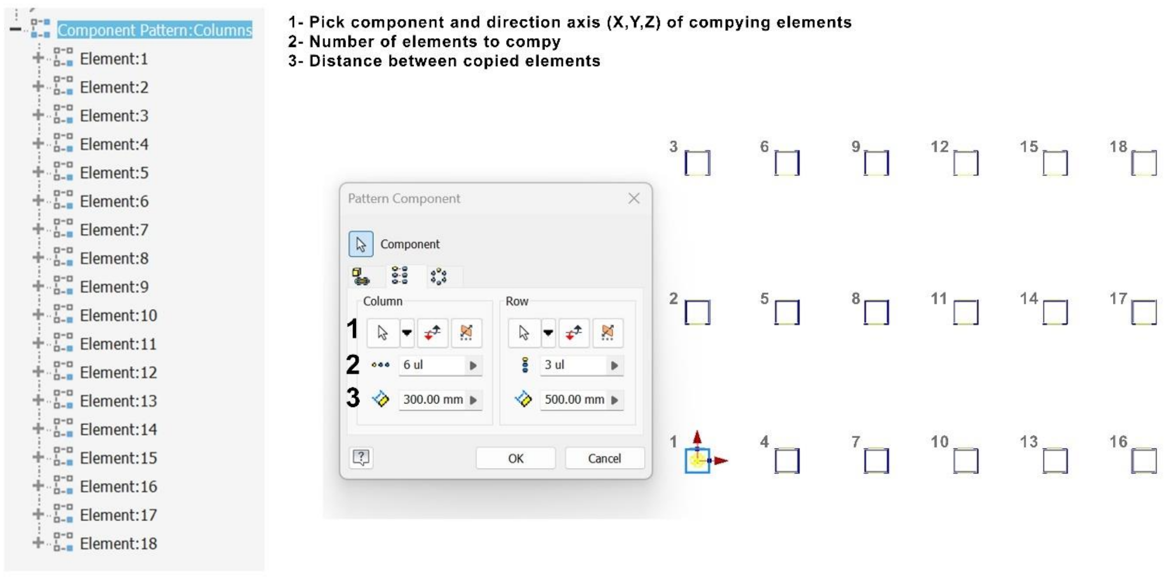
- Part names: PG808024700 – Pilar Galvanizado Quadrado 80x2.0mm

  -The P is From Pilar (Column);
  -The G is from Galvanizado (galvanized);
  -The 80802 are the measures of a column, a square of 80 x 80 mm with 2 mm of thickness.
  - 4700 presents height of column

- Parameters names: CubeBlueLength, words are written without space and with capital letters for every word.
- Rule names: Hole_Diameter, words are separated with underscore
- Folder names: Estufa9MFrontal, words are written without space and with capital letters for every word.

4.4. RECTANGULAR PATTERN

Steps from manual modelling are implemented in code to recreate a 3D model with software. *Rectangular pattern* command is used to manually create a model. In Inventor, this command is used to copy elements in the desired direction (X,Y,Z) with a defined number and distance between elements. Elements inside the pattern can be suppressed, except the first one. Each element is assigned with a number (see Figure 29). With that number, elements will be manipulated, suppressed or used as a starting point for the next pattern, etc.


1- Pick component and direction axis (X,Y,Z) of compying elements
2- Number of elements to compy
3- Distance between copied elements

Pattern Component — Component; Column: 6 ul, 300.00 mm; Row: 3 ul, 500.00 mm

Figure 29 – Pattern command explanation

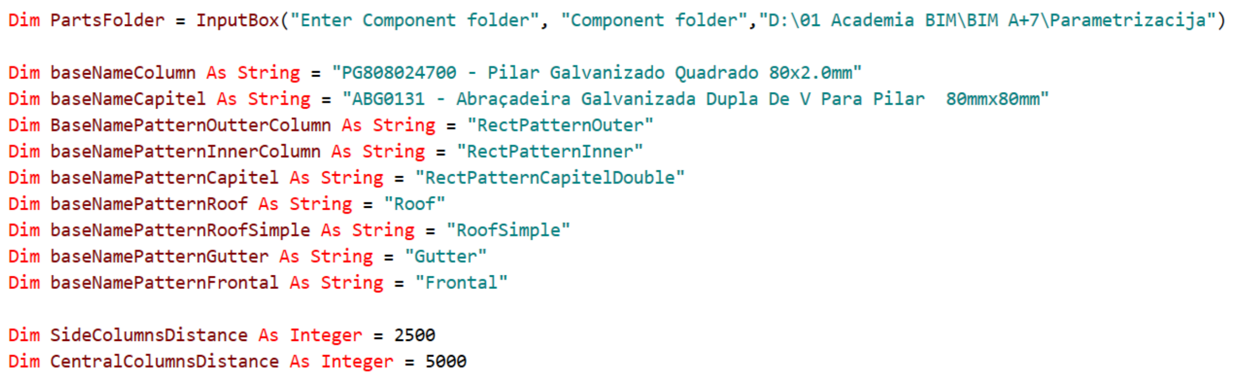
4.5. CODE

The main steps of the code are described in the following paragraphs.

1. **Class Definition:** The code starts by defining a class called GreenHouses (Line 7). In this case, the GreenHouses class serves as a container for organizing the code related to the assembly creation process. Class-level variables, like file paths and constants, are defined in class. Class-level variables can be accessed by any method within the class, which means that they can be used across different methods, without having to pass them as arguments.

```
Class GreenHouses
    ' Class-level variables

    Sub Main()
        ' Method code
        ' ...
    End Sub

    ' Other methods
    ' ...

End Class
```
(7)

2. **Variable Declarations:** Inside the class, various variables are declared (Figure 30). These variables hold information such as component and pattern names, dimensions, input values, and other parameters that will be used throughout the script.



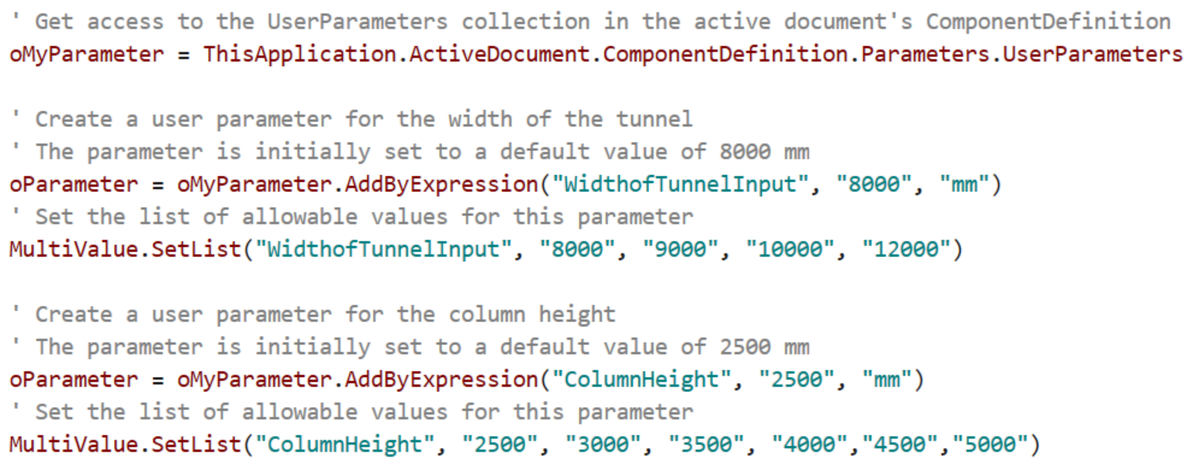
```
Dim PartsFolder = InputBox("Enter Component folder", "Component folder","D:\01 Academia BIM\BIM A+7\Parametrizacija")

Dim baseNameColumn As String = "PG808024700 - Pilar Galvanizado Quadrado 80x2.0mm"
Dim baseNameCapitel As String = "ABG0131 - Abraçadeira Galvanizada Dupla De V Para Pilar  80mmx80mm"
Dim BaseNamePatternOutterColumn As String = "RectPatternOuter"
Dim baseNamePatternInnerColumn As String = "RectPatternInner"
Dim baseNamePatternCapitel As String = "RectPatternCapitelDouble"
Dim baseNamePatternRoof As String = "Roof"
Dim baseNamePatternRoofSimple As String = "RoofSimple"
Dim baseNamePatternGutter As String = "Gutter"
Dim baseNamePatternFrontal As String = "Frontal"

Dim SideColumnsDistance As Integer = 2500
Dim CentralColumnsDistance As Integer = 5000
```

Figure 30 – Variable declaration

3. **Main Subroutine**: The Main subroutine is the entry point of the script where the assembly creation process starts. It involves user inputs, component creation, pattern generation, and more. (Line 8)

```
Sub Main()
    ' Code for user inputs, component creation, patterns,
etc.
    ' ...
End Sub
```
(8)

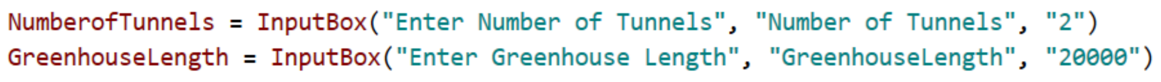
4. **Parameter creation:** This section creates user parameters for the assembly (Figure 31). For External rule too be able to run in every file, user parameters must be created for later use in code.



```
' Get access to the UserParameters collection in the active document's ComponentDefinition
oMyParameter = ThisApplication.ActiveDocument.ComponentDefinition.Parameters.UserParameters

' Create a user parameter for the width of the tunnel
' The parameter is initially set to a default value of 8000 mm
oParameter = oMyParameter.AddByExpression("WidthofTunnelInput", "8000", "mm")
' Set the list of allowable values for this parameter
MultiValue.SetList("WidthofTunnelInput", "8000", "9000", "10000", "12000")

' Create a user parameter for the column height
' The parameter is initially set to a default value of 2500 mm
oParameter = oMyParameter.AddByExpression("ColumnHeight", "2500", "mm")
' Set the list of allowable values for this parameter
MultiValue.SetList("ColumnHeight", "2500", "3000", "3500", "4000","4500","5000")
```

Figure 31 – Parameter creation

5. **User inputs:** User inputs, such as the width of the tunnel, number of tunnels, and greenhouse length are collected using InputListBox and InputBox functions. The values provided by users are stored in variables, as presented in Figure 32.

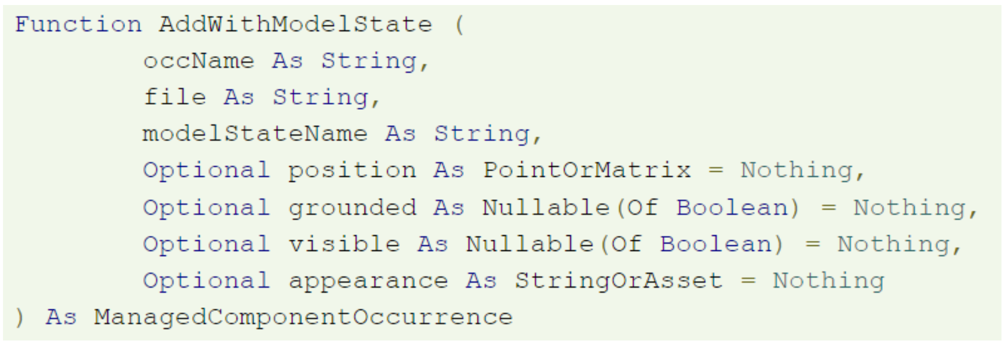


```
NumberofTunnels = InputBox("Enter Number of Tunnels", "Number of Tunnels", "2")
GreenhouseLength = InputBox("Enter Greenhouse Length", "GreenhouseLength", "20000")
```

Figure 32 – User inputs

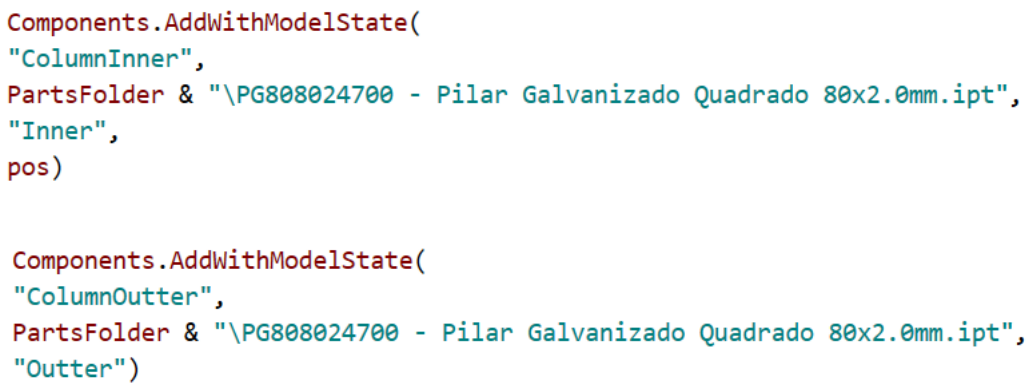
6. **Adding component:** Components such as outer columns, inner columns, capitel, roof, gutter, etc., are created and positioned in the assembly.

   a) The first way components are added is with a specified model state (**Figure 33**). The model state defines the variation of components. The first snippet is the explanation of the method, variables, and data type. The second and third are snippets for importing inner and outer columns. The inner column has a thickness of 2 mm, while the outer has a thickness of 3 mm) (Figure 34).



```
Function AddWithModelState (
        occName As String,
        file As String,
        modelStateName As String,
        Optional position As PointOrMatrix = Nothing,
        Optional grounded As Nullable(Of Boolean) = Nothing,
        Optional visible As Nullable(Of Boolean) = Nothing,
        Optional appearance As StringOrAsset = Nothing
) As ManagedComponentOccurrence
```

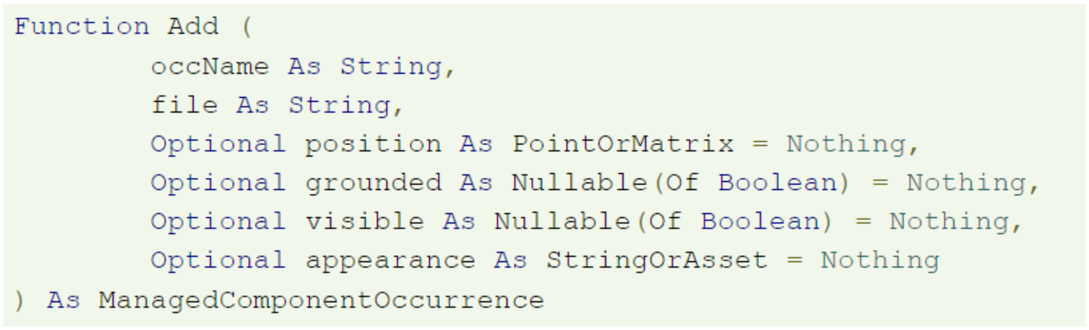
Figure 33 – API documentation for the AddWithModelState method



```
Components.AddWithModelState(
"ColumnInner",
PartsFolder & "\PG808024700 - Pilar Galvanizado Quadrado 80x2.0mm.ipt",
"Inner",
pos)

Components.AddWithModelState(
"ColumnOutter",
PartsFolder & "\PG808024700 - Pilar Galvanizado Quadrado 80x2.0mm.ipt",
"Outter")
```

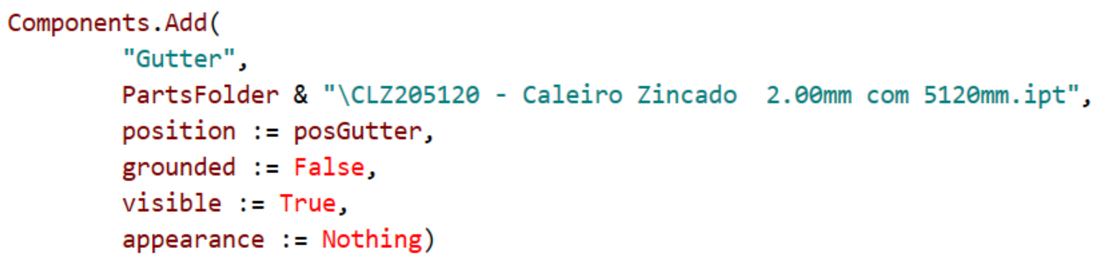
Figure 34 – AddWithModelState method implemented in case study code

   b) The second way to add components is with the Add method. The definition of this method in documentation is *Adds a component occurrence to an assembly or modify an existing component.* The first (Figure 35) snippet presents an explanation from documentation about variables and data type, while the second snippet (Figure 36) is the method implemented in the case study. Snippet is adding a gutter into the model space.



```
Function Add (
        occName As String,
        file As String,
        Optional position As PointOrMatrix = Nothing,
        Optional grounded As Nullable(Of Boolean) = Nothing,
        Optional visible As Nullable(Of Boolean) = Nothing,
        Optional appearance As StringOrAsset = Nothing
) As ManagedComponentOccurrence
```

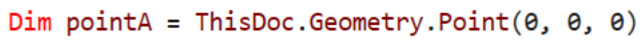
Figure 35 – API documentation for Add method



```
Components.Add(
        "Gutter",
        PartsFolder & "\CLZ205120 - Caleiro Zincado  2.00mm com 5120mm.ipt",
        position := posGutter,
        grounded := False,
        visible := True,
        appearance := Nothing)
```

Figure 36 – Add method implemented in case study code

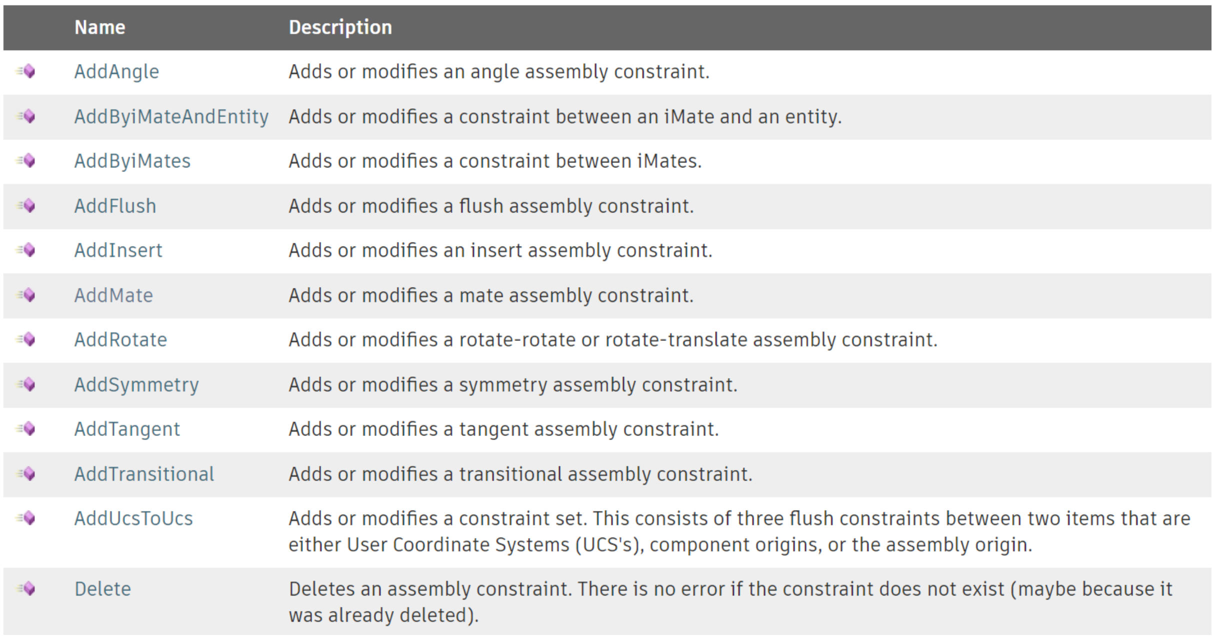
7. **Positioning the component within the main-assembly:** There are two ways a component is positioned within the main assembly.
   a) The first one with creation of the point with (X,Y,Z) coordinates and placing component in that point. (Figure 37)



```
Dim pointA = ThisDoc.Geometry.Point(0, 0, 0)
```

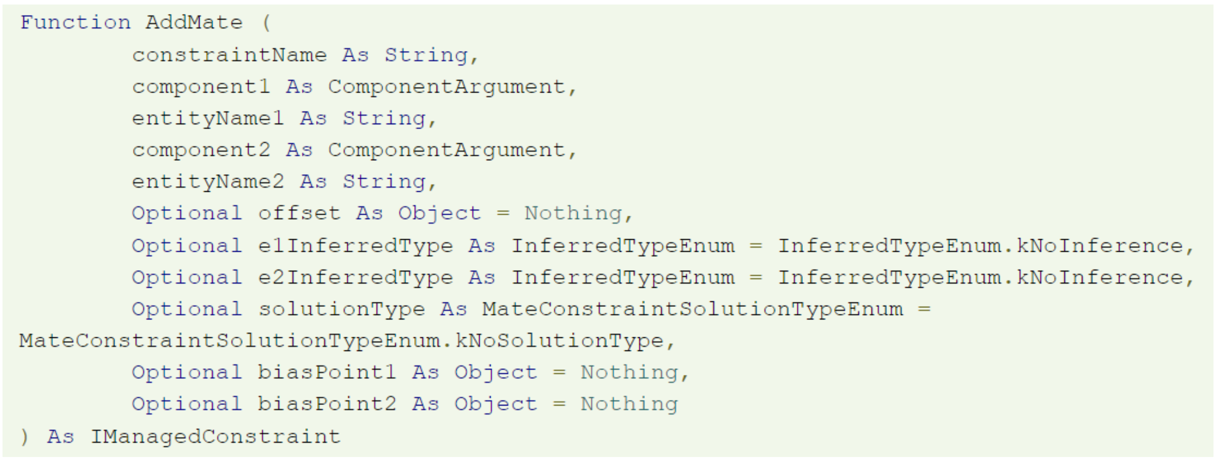
Figure 37 – Snipet of code for point creation

   b) The second way of positioning components is with constraints. As is visible in Figure 38 all constraints available manually in Inventor are also available programmatically through Inventor API. iLogic offers predefined snippets for these constraints which makes the process easier and more intuitive. Figure 39 shows API documentation for Mate constraints one used in the case study, while Figure 40 presents its implementation inside the code.



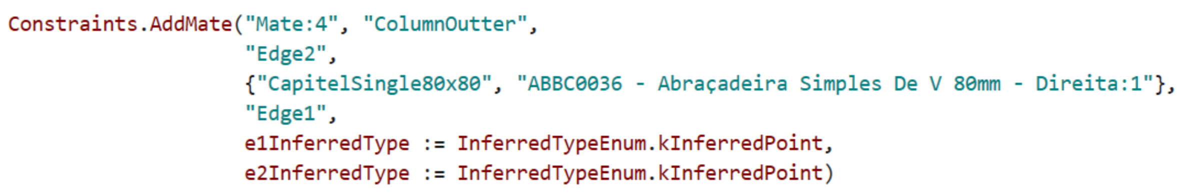
Name	Description
AddAngle	Adds or modifies an angle assembly constraint.
AddByiMateAndEntity	Adds or modifies a constraint between an iMate and an entity.
AddByiMates	Adds or modifies a constraint between iMates.
AddFlush	Adds or modifies a flush assembly constraint.
AddInsert	Adds or modifies an insert assembly constraint.
AddMate	Adds or modifies a mate assembly constraint.
AddRotate	Adds or modifies a rotate-rotate or rotate-translate assembly constraint.
AddSymmetry	Adds or modifies a symmetry assembly constraint.
AddTangent	Adds or modifies a tangent assembly constraint.
AddTransitional	Adds or modifies a transitional assembly constraint.
AddUcsToUcs	Adds or modifies a constraint set. This consists of three flush constraints between two items that are either User Coordinate Systems (UCS's), component origins, or the assembly origin.
Delete	Deletes an assembly constraint. There is no error if the constraint does not exist (maybe because it was already deleted).

Figure 38 – Methods available within the IManagedConstraints interface



```
Function AddMate (
        constraintName As String,
        component1 As ComponentArgument,
        entityName1 As String,
        component2 As ComponentArgument,
        entityName2 As String,
        Optional offset As Object = Nothing,
        Optional e1InferredType As InferredTypeEnum = InferredTypeEnum.kNoInference,
        Optional e2InferredType As InferredTypeEnum = InferredTypeEnum.kNoInference,
        Optional solutionType As MateConstraintSolutionTypeEnum =
MateConstraintSolutionTypeEnum.kNoSolutionType,
        Optional biasPoint1 As Object = Nothing,
        Optional biasPoint2 As Object = Nothing
) As IManagedConstraint
```

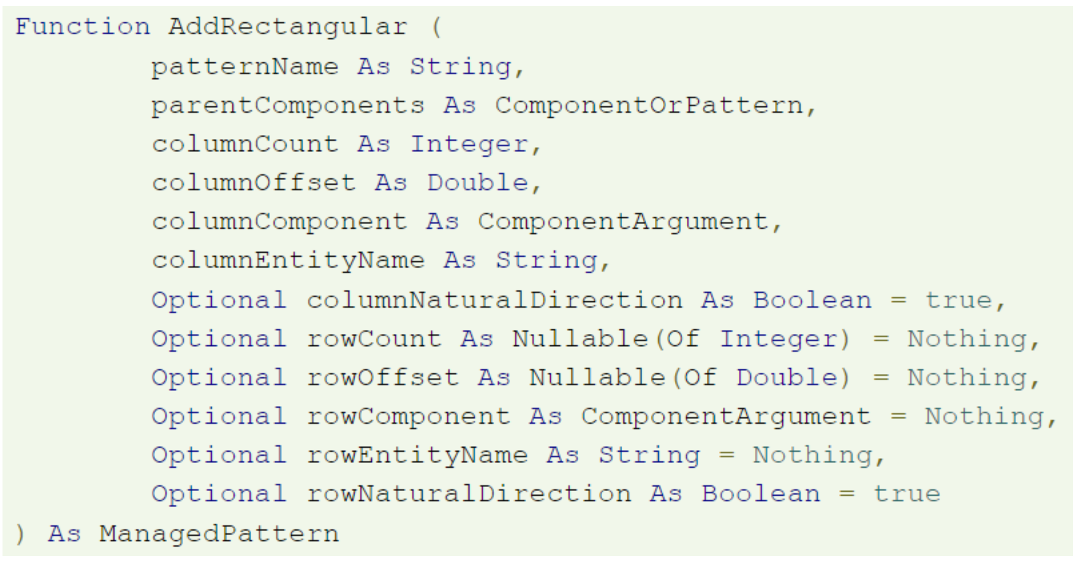
Figure 39 – API documentation for the AddMate method



```
Constraints.AddMate("Mate:4", "ColumnOutter",
                    "Edge2",
                    {"CapitelSingle80x80", "ABBC0036 - Abraçadeira Simples De V 80mm - Direita:1"},
                    "Edge1",
                    e1InferredType := InferredTypeEnum.kInferredPoint,
                    e2InferredType := InferredTypeEnum.kInferredPoint)
```

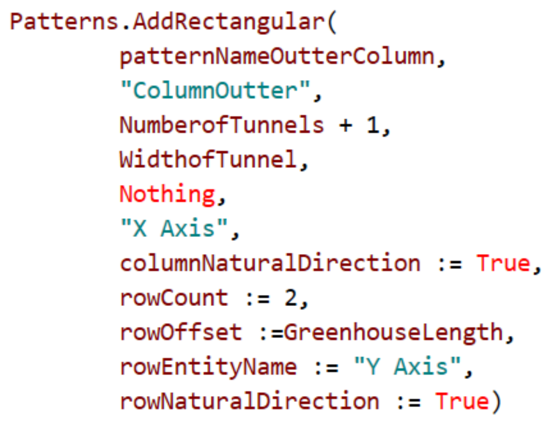
Figure 40 – Add method implemented in case study code

8. **Pattern Creation:** Patterns are created for various components like outer columns, inner columns, capitel, roof, etc. Within the documentation, this method is described as *Adds or modifies a rectangular pattern in an assembly*. Figure 41 presents API explanation of this method. The second snippet (Figure 42) explains how outer columns are copied into the pattern, in the X direction it is NumberofTunnels + 1 times with the distance of WidthofTunnel while in the Y direction, 2 columns are made with the distance of GreenhouseLength.



```
Function AddRectangular (
        patternName As String,
        parentComponents As ComponentOrPattern,
        columnCount As Integer,
        columnOffset As Double,
        columnComponent As ComponentArgument,
        columnEntityName As String,
        Optional columnNaturalDirection As Boolean = true,
        Optional rowCount As Nullable(Of Integer) = Nothing,
        Optional rowOffset As Nullable(Of Double) = Nothing,
        Optional rowComponent As ComponentArgument = Nothing,
        Optional rowEntityName As String = Nothing,
        Optional rowNaturalDirection As Boolean = true
) As ManagedPattern
```

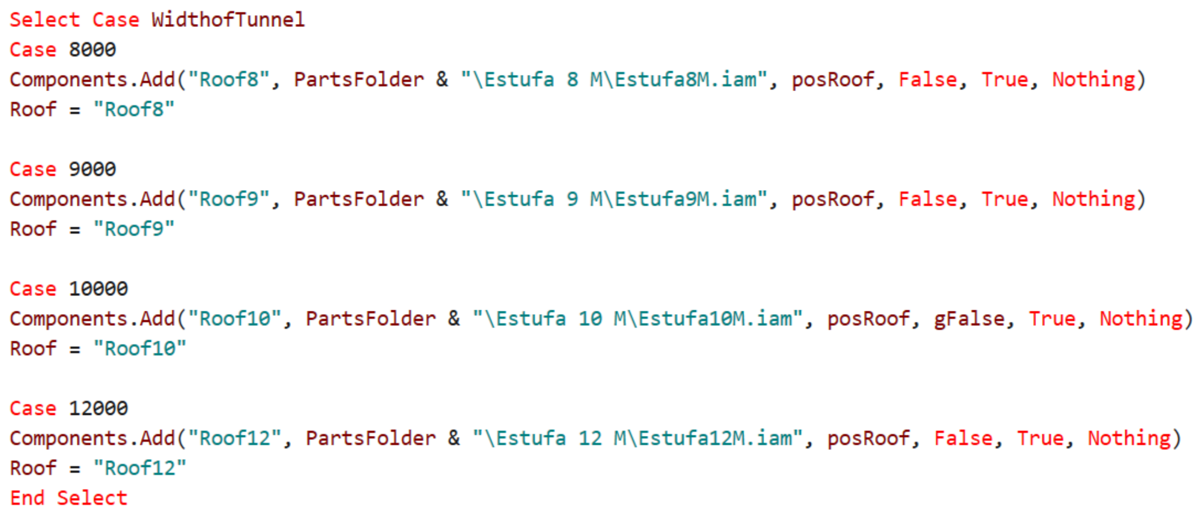
Figure 41 – API documentation for AddRectangular method



```
Patterns.AddRectangular(
        patternNameOutterColumn,
        "ColumnOutter",
        NumberofTunnels + 1,
        WidthofTunnel,
        Nothing,
        "X Axis",
        columnNaturalDirection := True,
        rowCount := 2,
        rowOffset :=GreenhouseLength,
        rowEntityName := "Y Axis",
        rowNaturalDirection := True)
```

Figure 42 – AddRectangular method implemented in case study code

9. **Condition-Based Component Adding:** Different versions of sub-assemblies (e.g., roof, frontal) are selected and added, based on the chosen width of the greenhouse. Figure 43 shows how roof subassembly is imported based on the chosen width of tunnel.



```
Select Case WidthofTunnel
Case 8000
Components.Add("Roof8", PartsFolder & "\Estufa 8 M\Estufa8M.iam", posRoof, False, True, Nothing)
Roof = "Roof8"

Case 9000
Components.Add("Roof9", PartsFolder & "\Estufa 9 M\Estufa9M.iam", posRoof, False, True, Nothing)
Roof = "Roof9"

Case 10000
Components.Add("Roof10", PartsFolder & "\Estufa 10 M\Estufa10M.iam", posRoof, gFalse, True, Nothing)
Roof = "Roof10"

Case 12000
Components.Add("Roof12", PartsFolder & "\Estufa 12 M\Estufa12M.iam", posRoof, False, True, Nothing)
Roof = "Roof12"
End Select
```

Figure 43 – Selection of roof based on the chosen width of the tunnel functionality

The sequence in which components are being added and patterns are being created is represented in Figure 44 and described in the following lines:

1. The creation of geometry starts by importing the model state for the outer column and setting its parameters (column section and height). After defining the outer column, pattern is created.
2. The script then moves on to work with inner columns, by taking the proper model state of the column that is used for the inner column (Thickness=2), after which a pattern is created.
3. The next steps are importing double capitel and creating patterns.
4. The right component for the roof structure is added, based on the user's input Width of the tunnel, and creates patterns for the roof structure for every odd number, starting from the 3rd column.
5. Every even column starting from second, is filled with a simplified version of the roof.
6. A similar process is done for the frontal and end side of the greenhouse, the right subassembly is imported, and a pattern is created.
7. The following element that is added is Single capitel positioned on the top of the column and for the first line of columns capitels are created.
8. Following element is gutter that is constrained to capitel after wich pattern is created.
9. First and last line of columns in Y direction is created, but before that column is imported in right position.
10. The next elements added are clamps. They are adjusted according to the ColumnSection and fixed to the column in the right position. The number of clamps is defined according to the Height of the column and a pattern is created.
11. The following components are horizontal profiles. They are fixed with clamps and made into a pattern that is defined by the Column Height.

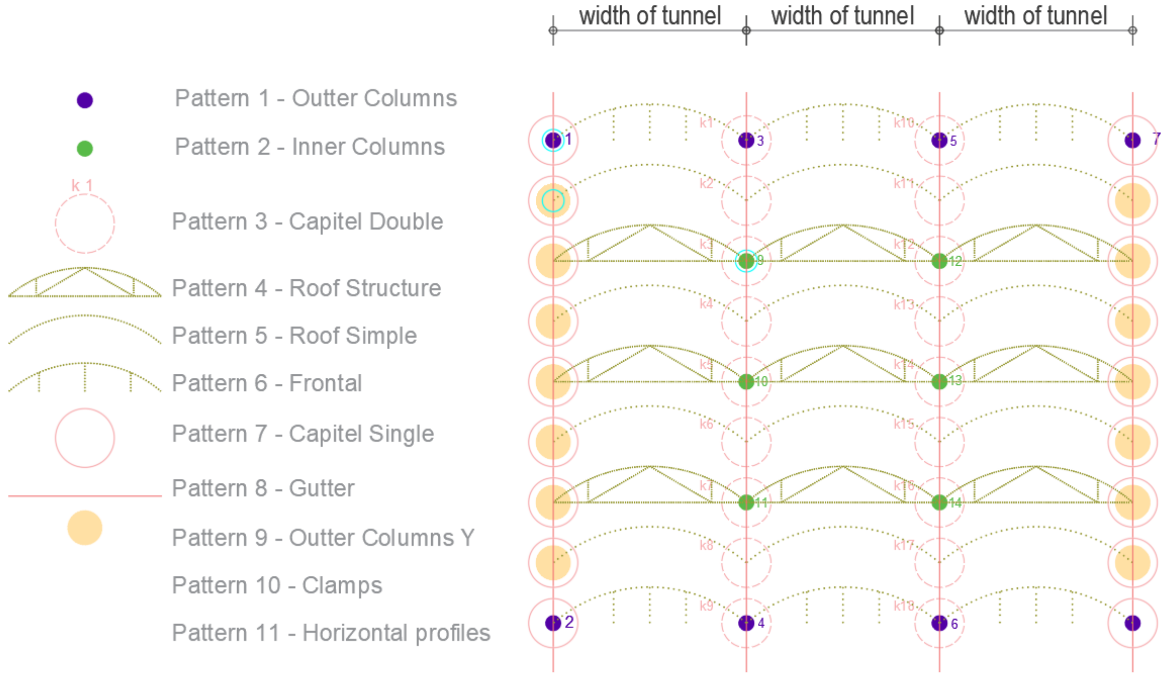

width of tunnel
width of tunnel
width of tunnel
Pattern 1 - Outter Columns
Pattern 2 - Inner Columns
k 1
Pattern 3 - Capitel Double
Pattern 4 - Roof Structure
Pattern 5 - Roof Simple
Pattern 6 - Frontal
Pattern 7 - Capitel Single
Pattern 8 - Gutter
Pattern 9 - Outter Columns Y
Pattern 10 - Clamps
Pattern 11 - Horizontal profiles

Figure 44 – Sequence of patterns creation

4.6. STEPS FOR IMPORTING NEW GREENHOUSE ELEMENTS

For implementation of new elements these are following steps that are suggested to simplify and systematize the process.

Step 1: Analyze Needs and Requirements: Analyze needs and requirements and define functionality that will satisfy them. For this example, functionality that can be implemented is ventilation.

Step 2: Capture Technical Solutions: Explore different technical solutions for providing ventilation in greenhouses. This could involve options like roof ventilation, ridge ventilation, and side ventilation.

Step 3: Decompose Technical Solution into Modules: Break down the ventilation solution into modular components. What will be considered a module or component depends on the model's complexity and if it has to support variation.

Step 4: Analyze Module Variants: Examine variations within each module. Document these variants.

Step 5: Define Influencing Functionalities: Identify the functionalities that influence each ventilation module. For example, the functionality greenhouse length will influence the length of ventilation, etc.

Step 6: Systematize Component Knowledge: Compile detailed information about each module, including properties, dimensions, etc.

Step 7: Analyze Geometry and define the way it will be Imported: Determine how the geometry of each ventilation module will be imported into a main-assambly. Decide whether it will be imported as a sub-assembly, individual part, or model state.

Step 8: Positioning and Placement: Analyze where each ventilation module should be placed within the original file, how it should be related to the origin planes, and how should be related to other elements in the main assembly file.

Step 9: Define Pattern: If multiple instances of a ventilation module are required, define a pattern for their placement. Specify variables that determine the distance between instances and their number.

Step 10: Incorporate in Code: Translate the design and placement specifications into code..

4.7. USE CASE

The program will be used by engineers to generate a greenhouse model. To run code, it is necessary to run the iLogic add-in and in the section External Rules "Add External Rule" from the location where it is saved. To use the Rule, it is necessary to run it inside the .iam template, the file template should be in millimetres (Figure 45).

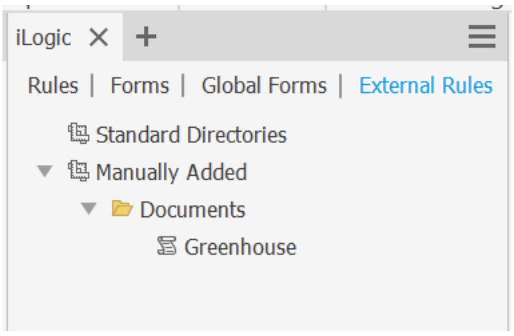
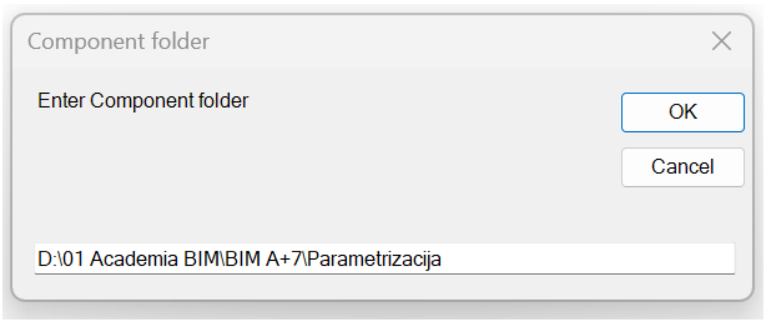


Figure 45 – External Rules window inside Inventor

The width can be selected from the available options: 8000 mm, 9000 mm, 10000 mm, and 12000 mm. The resulting greenhouse structure is composed of 8 elements, with the possibility of adjustments, according to the company's needs.

For a program to work, it has to make a connection to a database. That connection is established by filling the input box with a link towards the folder location (Figure 46).



Component folder

Enter Component folder

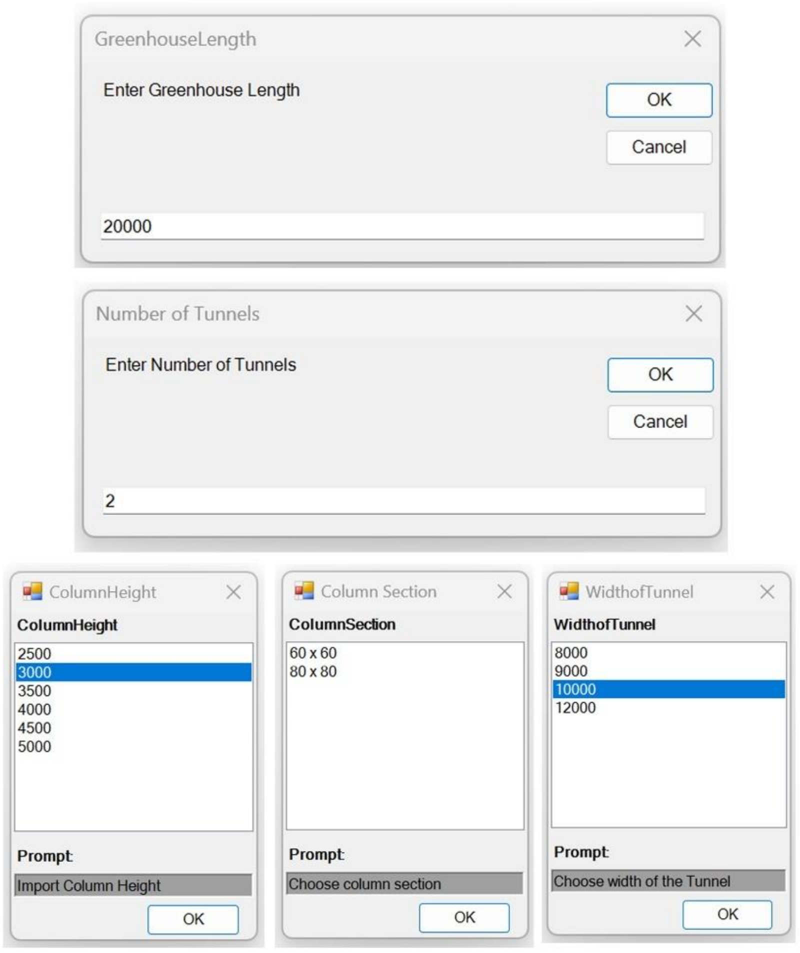
OK

Cancel

D:\01 Academia BIM\BIM A+7\Parametrizacija

Figure 46 – Input window for link towards the components folder

To initiate the process, the user defines inputs through dedicated input forms. These inputs are the tunnel width, the number of tunnels, greenhouse length, column height, and column section (Figure 47 The user's selections shapes the greenhouse's configuration. Upon completing the input stage, the program employs the defined parameters to generate a detailed model of the greenhouse. This model provides insights into the greenhouse's layout, design, and specifications. Furthermore, the program's interface ensures a seamless and intuitive input process.



GreenhouseLength

Enter Greenhouse Length

OK

Cancel

20000

Number of Tunnels

Enter Number of Tunnels

OK

Cancel

2

ColumnHeight

ColumnHeight

2500
3000
3500
4000
4500
5000

Prompt:

Import Column Height

OK

Column Section

ColumnSection

60 x 60
80 x 80

Prompt:

Choose column section

OK

WidthofTunnel

WidthofTunnel

8000
9000
10000
12000

Prompt:

Choose width of the Tunnel

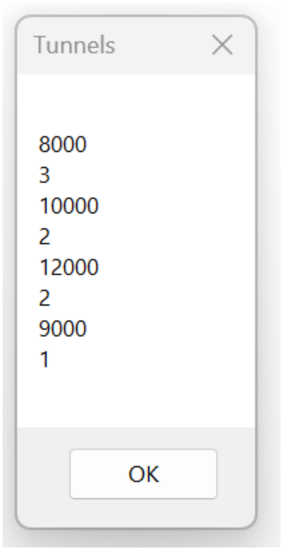
OK

Figure 47 – Input windows

4.8. IMPLEMENTATION OF MULTIPLE TUNNEL WIDTHS

The logic of the manually created model was implemented in the code, and the idea is to replicate the process. After establishing code that works for one tunnel width, the idea is to incorporate a combination of two, three, and more widths in one greenhouse structure, and therefore, one greenhouse can have different width combinations. As in the previous case, rectangular pattern command is also used here. Due to limited time this code will be elaborated only with columns.

In the first step, the user inputs the width of the tunnels and the number of tunnels. These values are stored within the list that will be later used for pattern naming, pattern creation, etc (Figure 48). The first index of the list is 0. The list is structured in a way that even indexes of the list present the width of the tunnel and odd indexes represent the number of tunnels. After values with indexes 0 and 1 are used to create patterns of inner and outer columns, values are deleted, and these indexes are assigned with the next 2 values from the list. The code will iterate as long as there are items in the list. Values that are stored in the list and needed in the code are always referred to with the index 0 and 1. List (0) is equal to the width of the tunnel and List (1) is equal to the number of tunnels.



```
8000
3
10000
2
12000
2
9000
1
```

Figure 48 – List of stored values of greenhouse width and number of tunnels

Depending on how many inputs are in the list, code will be executing different code groups.

1. If there are 2 inputs in the list (list.Count=2), meaning that there is only one width in the greenhouse structure, the following code groups are called:
   - First pattern group (code will make inner and outer columns)
   - Y columns (code will make 1st and last group of columns in Y direction)

2. If there are 4 inputs in the list (list.Count=4), the greenhouse consists of 2 groups of tunnel widths, and following code is called:
   - First pattern group (code will make inner and outer columns)
   - Suppress columns (in overlapping zone code will suppress duplicated columns)

- The second pattern group (code will make inner and outer columns with second widths from inputs
- Y columns

3. If there are 6 or more inputs in the list (list.count>=6), it means that the greenhouse consists of a combination of 3 or more tunnel widths, and the following code is executed:
   - First Pattern group
   - Suppress columns
   - Second Pattern group
   - Suppress columns
   - Third Pattern group
   - Y columns

The following steps are taken to create the Second and Third groups of patterns:

1) When there is only one width in the greenhouse, the creation of the model is straightforward. Firstly, an outer column is imported in coordinate (0,0,0) then a pattern is created. Secondly, the inner column is imported in coordinate (WidthofTunnel,5000,0) and a pattern is created, thirdly, the column is imported on position (0,2500,0) pattern for 1st and last line of columns in Y direction is created. Figure 49 shows this process.

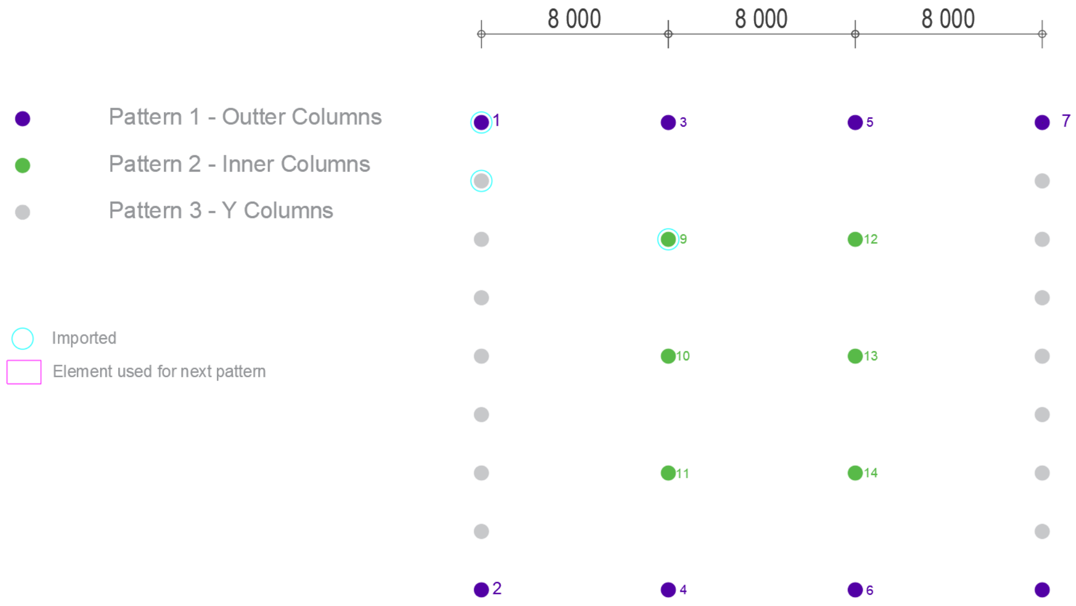

8 000
8 000
8 000
Pattern 1 - Outter Columns
Pattern 2 - Inner Columns
Pattern 3 - Y Columns
Imported
Element used for next pattern
1
3
5
7
9
12
10
13
11
14
2
4
6

Figure 49 – Column patterns for greenhouse with one width

2) The second case is a greenhouse with a combination of 2 different tunnel widths. The logic will be demonstrated in the example from Figure 50.The inner and outer patterns with the first width will be executed as previously described. The previous outer pattern (Pattern 1) is called and column number 7 is made independent. The column is called with the use of the formula CurrentIndexColumn = 1 + (list(1) * 2). When the column is independent, it is used for the next outer column pattern (Pattern 3). The same is done for the previous pattern inner, column number 15 (CurrentIndexColumnInner = CurrentIndexColumn +1 + CentralColumns

* (list(1)-1) + 1) is called and pattern number 4 is created. From Pattern number 3 number 19 is made independent and used for the creation of a pattern of outer columns in the Y direction. The zone between these two width groups has columns from both pattern widths, so columns are duplicated. To avoid duplicated elements, columns are suppressed. **Table 3 – Pattern creation for two-width combination greenhouse** the sequence of pattern creation, formulas that are calling the right columns within the pattern, conditions for pattern creation, and the name of the code that is being called.

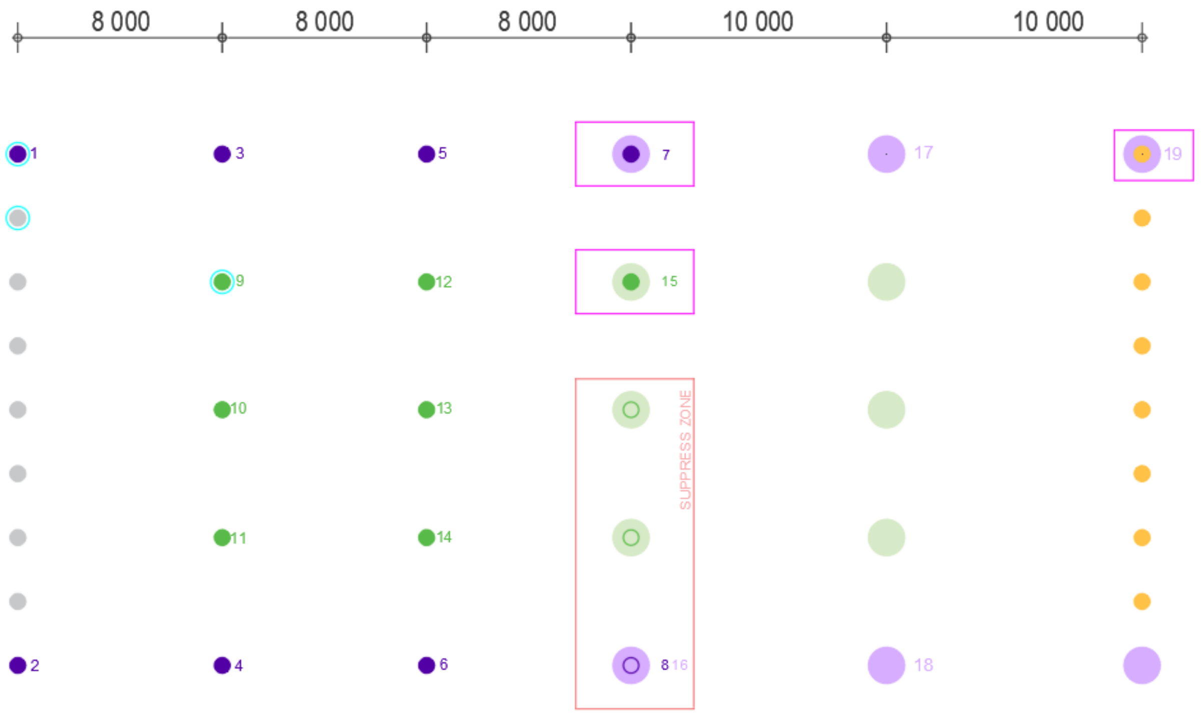
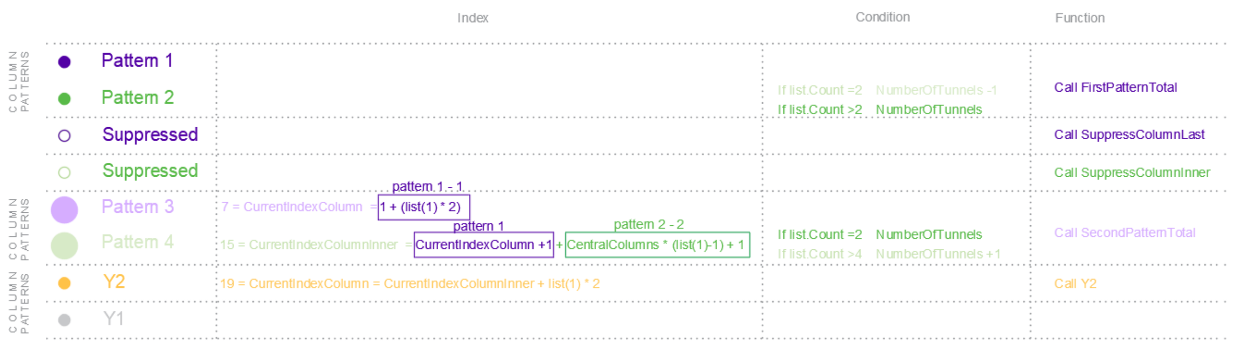


Figure 50 – Column patterns for greenhouse with two widths

Table 3 – Pattern creation for two-width combination greenhouse



		Index	Condition	Function
COLUMN PATTERNS	● Pattern 1 ● Pattern 2		If list.Count =2 NumberOfTunnels -1 If list.Count >2 NumberOfTunnels	Call FirstPatternTotal
	○ Suppressed			Call SuppressColumnLast
	○ Suppressed			Call SuppressColumnInner
COLUMN PATTERNS	● Pattern 3 ● Pattern 4	7 = CurrentIndexColumn = 1 + (list(1) * 2) (pattern 1 - 1) 15 = CurrentIndexColumnInner = CurrentIndexColumn +1 (pattern 1) + CentralColumns * (list(1)-1) + 1 (pattern 2 - 2)	If list.Count =2 NumberOfTunnels If list.Count >4 NumberOfTunnels +1	Call SecondPatternTotal
COLUMN PATTERNS	● Y2	19 = CurrentIndexColumn = CurrentIndexColumnInner + list(1) * 2		Call Y2
	● Y1			

The diagram presented in Figure 51 shows the sequence of actions that are needed to generate groups of patterns that will be made with a second width. The first step is to define the column using an index from the previous tunnel width, in this case, it is value 3. Secondly, the variable name is defined and that will be used to store column and to make a pattern. The following step is calling the pattern with a previously defined name using index 0 that has a value of 8000, so the pattern name is in this case RecPattenOutter8000 from where the column will be made independant. In the following step values 8000 and 3 are deleted and new values for list(0)=10000 and list(1)=2. With these values, it is defined pattern name that is created in the next step.

MessageBox.Show("Message", "Title") is a function that is used as one of the debugging strategies to follow variables and their values throughout the code. Code that resembles these actions is presented in Figure 52

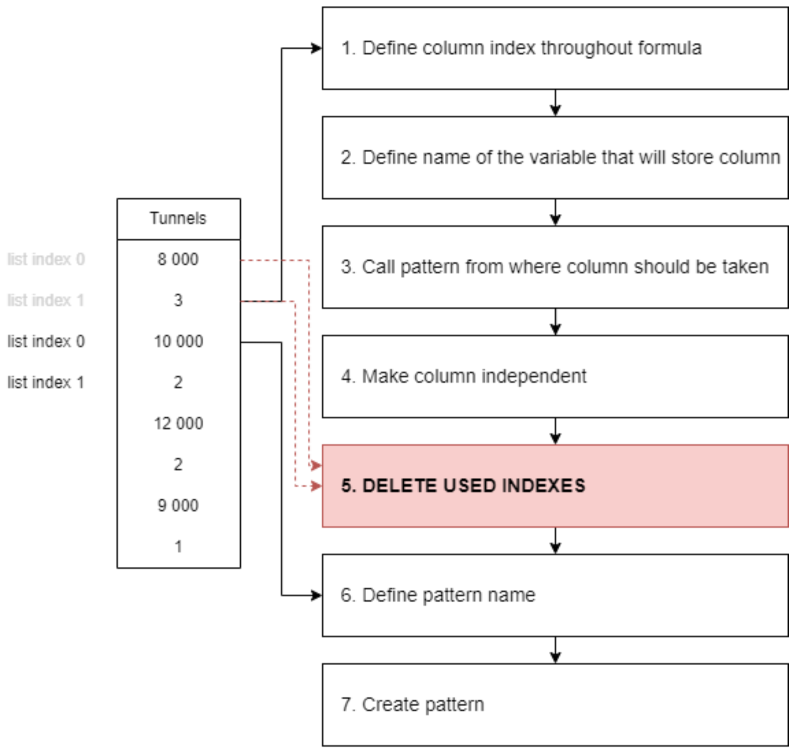
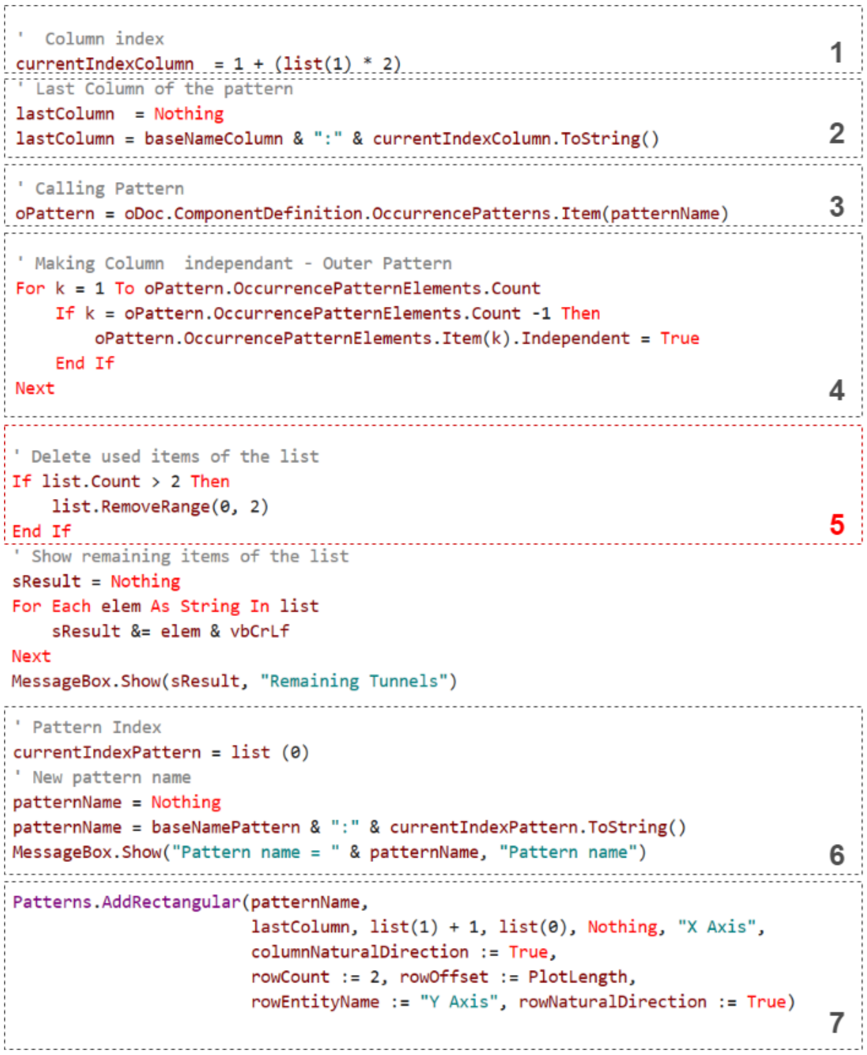

1. Define column index throughout formula
2. Define name of the variable that will store column
3. Call pattern from where column should be taken
4. Make column independent
5. DELETE USED INDEXES
6. Define pattern name
7. Create pattern
Tunnels
8 000
3
10 000
2
12 000
2
9 000
1
list index 0
list index 1
list index 0
list index 1

Figure 51 – Sequence of actions in code to create the pattern with the right name assigned



```
'  Column index
currentIndexColumn  = 1 + (list(1) * 2)                                                    1
' Last Column of the pattern
lastColumn  = Nothing
lastColumn = baseNameColumn & ":" & currentIndexColumn.ToString()                          2
' Calling Pattern
oPattern = oDoc.ComponentDefinition.OccurrencePatterns.Item(patternName)                   3
' Making Column  independant - Outer Pattern
For k = 1 To oPattern.OccurrencePatternElements.Count
    If k = oPattern.OccurrencePatternElements.Count -1 Then
        oPattern.OccurrencePatternElements.Item(k).Independent = True
    End If
Next                                                                                       4
' Delete used items of the list
If list.Count > 2 Then
    list.RemoveRange(0, 2)
End If                                                                                     5
' Show remaining items of the list
sResult = Nothing
For Each elem As String In list
    sResult &= elem & vbCrLf
Next
MessageBox.Show(sResult, "Remaining Tunnels")
' Pattern Index
currentIndexPattern = list (0)
' New pattern name
patternName = Nothing
patternName = baseNamePattern & ":" & currentIndexPattern.ToString()
MessageBox.Show("Pattern name = " & patternName, "Pattern name")                           6
Patterns.AddRectangular(patternName,
                        lastColumn, list(1) + 1, list(0), Nothing, "X Axis",
                        columnNaturalDirection := True,
                        rowCount := 2, rowOffset := PlotLength,
                        rowEntityName := "Y Axis", rowNaturalDirection := True)
                                                                                           7
```

Figure 52 – Code for creation of outer column pattern

3) The third pattern group that represents a 3rd width, which is done the same way as the second. Columns that are used for pattern creation are called, just this time, with different column indexes (column numbers in this case are 19 and 26). For every iteration after the 3rd, formulas will be the same as in the 3rd. This code will continue to iterate until there are no elements on the list anymore. Figure 53 shows patterns when there are three tunnel widths within the greenhouse structure. **Table 4** shows the sequence of pattern creation, formulas that are used for calling the right column and conditions that need to be satisfied.

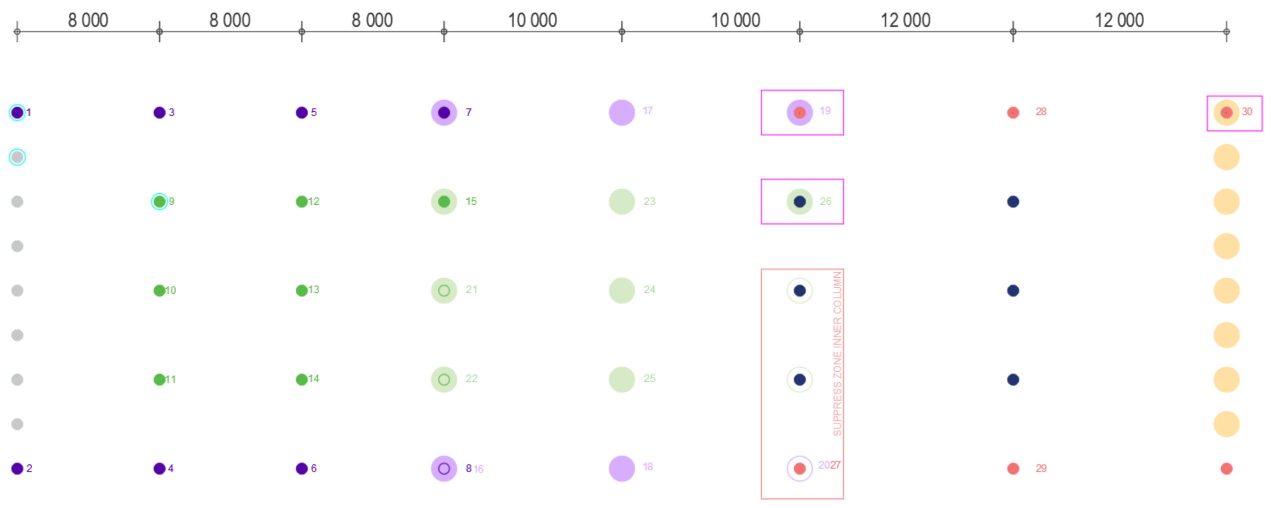


Figure 53 – Column patterns for greenhouse with three widths

Table 4 – Pattern creation for three-width combination greenhouse

	Index	Condition	Function
● Pattern 1 ● Pattern 2		If list.Count =2 NumberOfTunnels -1 If list.Count >2 NumberOfTunnels	Call FirstPatternTotal
○ Suppressed			Call SuppressColumnLast
○ Suppressed			Call SuppressColumnInner
● Pattern 3 ● Pattern 4	7 = CurrentIndexColumn = $\overset{\text{pattern 1 - 1}}{\boxed{1 + (list(1) * 2)}}$ 15 = CurrentIndexColumnInner = $\overset{\text{pattern 1}}{\boxed{CurrentIndexColumn +1}}$ + $\overset{\text{pattern 2 - CC}}{\boxed{CentralColumns * (list(1)-1) + 1}}$	If list.Count =2 NumberOfTunnels -1 If list.Count >4 NumberOfTunnels	Call SecondPatternTotal
○ Suppressed			Call SuppressColumnLast
○ Suppressed			Call SuppressColumnInner
● Pattern 5 ● Pattern 6	19 = CurrentIndexColumn = CurrentIndexColumnInner + $\overset{\text{pattern 3}}{\boxed{list(1) * 2}}$ 26 = CurrentIndexColumnInner = $\overset{\text{pattern 3}}{\boxed{CurrentIndexColumn +1}}$ + $\overset{\text{pattern 4 - CC}}{\boxed{CentralColumns * list(1)}}$	If list.Count =2 NumberOfTunnels -1 If list.Count >4 NumberOfTunnels	Call ThirdPatternTotal
● Y2	19 = CurrentIndexColumn = CurrentIndexColumnInner + list(1) * 2		Call Y2
● Y1			

Figure 54 shows a column grid with 4 width combinations, that was created with the previously explained code. Figure 55 shows the naming of the patterns generated through code.

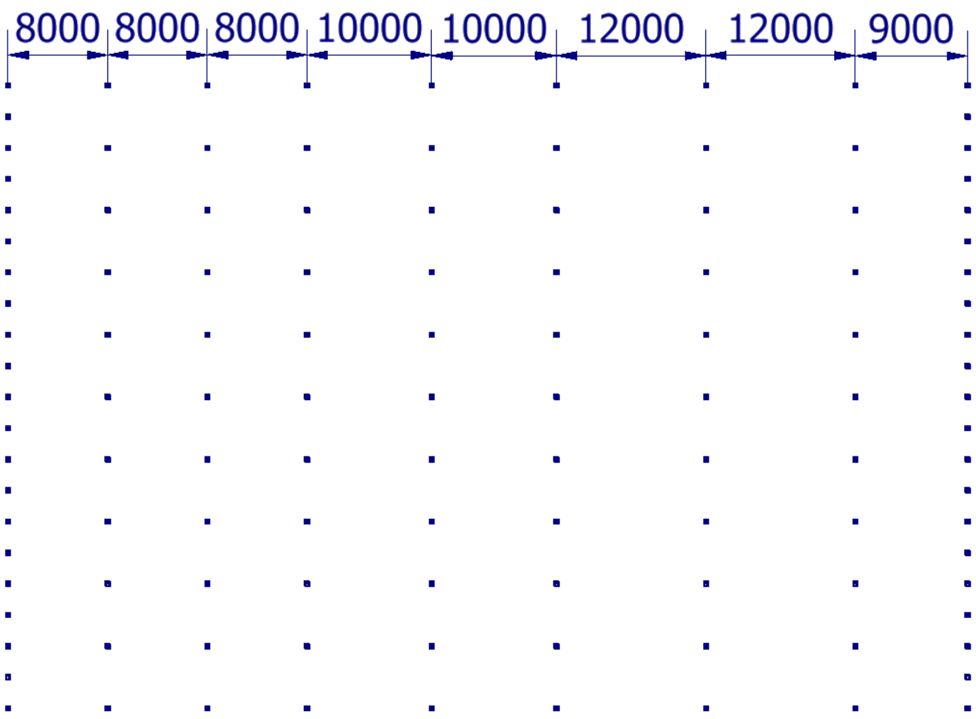


Figure 54 – 3D model of column grid with 4 tunnel widths combined into one greenhouse

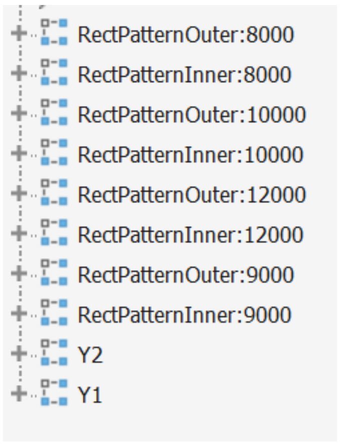

RectPatternOuter:8000
RectPatternInner:8000
RectPatternOuter:10000
RectPatternInner:10000
RectPatternOuter:12000
RectPatternInner:12000
RectPatternOuter:9000
RectPatternInner:9000
Y2
Y1

Figure 55 – Pattern naming for the 3D model from Figure 54

This page is intentionally left blank

5. DISCUSSION

5.1. GENERAL OVERVIEW

The main objective of this research was to explore the potential of parametrization and automatization of tasks in the context of greenhouse design, using Autodesk Inventor. The literature review was conducted with two main goals: to find similar research and projects that were implemented in Inventor and to discuss the main theoretical concepts that were applied in this work.

During the literature review, it was not possible to find projects that apply parametrization in Inventor for the assembly generation of complex structures, such as greenhouses. This reality shows the need to bridge this gap by exploring Inventor's functionalities, programming concepts, VB.NET, and Inventor API to automate the generation of greenhouse assemblies.

To gain insight into the practical application of parametric design within the context of greenhouse structures, the study initiated an analysis of the greenhouse structure itself. This analysis included an examination of greenhouse modules, components, and variations. Additionally, it was explored how customer inputs influence the configuration of the greenhouse. The complex structure and parameter relations were captured and systematically organized using diagrams.

For easier development of the code and implementation of core concepts, the structure was reduced to eight exemplary elements.

The study proceeded to develop a systematic approach for handling variations of parts and sub-assemblies, which results in three distinct options. The first option involves the direct importation of sub-assemblies, the second option triggers a rule within a part that governs its parameters. The third option revolves around the regulation of dimensions through the utilization of model states.

The code development process involved analyzing manually created models and identifying repetitive patterns, particularly the use of the rectangular pattern command for creating grids of elements. This manual creation process served as a model for the code development phase. By accessing functionalities through Inventor API, the manual modelling steps were translated into the code. The implemented logic worked effectively, producing expected outcomes.

5.2. CODE OUTPUTS

The external rule that automatically generates the greenhouse assembly creates a model within a few seconds. That significantly reduces time compared to the manual creation of the model. Minimum number of tunnels is 2. It consists of eight elements with the possibility of importing more and forming the full structure of a greenhouse. Figure 56 shows a 3D model generated by the code with the following inputs: width of the tunnel = 8000 mm, number of tunnels = 6, column height = 3000 mm, column section = 80×80.

Some values are taken as constants, like the distance between outer columns in the Y direction with a value of 2500 mm, and the distance between central columns with a value of 5000 mm. Code will produce outputs for these constant values. Exceptions will require additional manual corrections. Figure 57 shows a greenhouse with a tunnel length of 12125mm. The length is not divisible by 2500 which creates a geometry that requires manual adjustments.

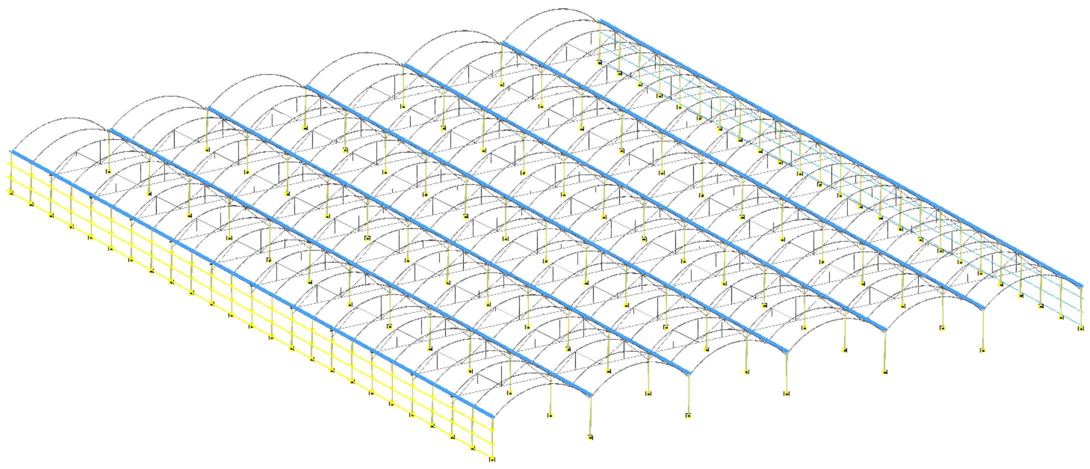


Figure 56 – Code output

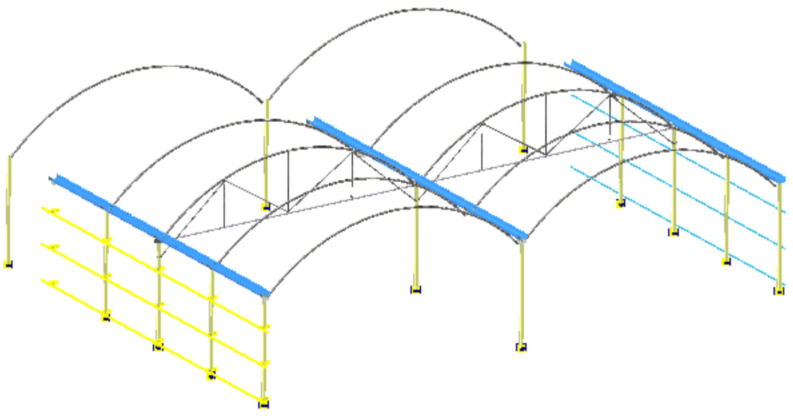


Figure 57 – Code output with non-typical inputs

5.3. INTERPRETATIONS AND IMPLEMENTATION OF RESULTS

The successful implementation of the code highlights the potential for automation in greenhouse design using parametrization and automated assembly generation. By translating the manual design process into code, the study demonstrated the feasibility of using programming concepts to streamline and expedite the creation of complex structures like greenhouses. The adoption of parametric design methodologies, coupled with programming tools, has the potential to significantly reduce the time and effort required for design tasks. Within a few seconds based on the user's inputs, the program generates a 3D model.

The utilization of parametric design as an automation tool can also lead to more efficient and accurate design processes, allowing designers to focus on critical decision-making aspects and higher value-added tasks. This can ultimately contribute to enhanced productivity, reduced manual errors, and improved overall consistency in greenhouse design. The successful integration of modern technologies into traditional design practices showcases the potential for technology-driven solutions to address problems of low productivity in the construction sector.

5.4. RESEARCH LIMITATIONS AND DIFFICULTIES

One limitation of this research is that the complexity of the code is increased with the introduction of multiple tunnel widths, raising questions about code modularity and maintainability. Furthermore, the study elaborated code logic of multiple-width tunnel structure, however, it did not explore potential challenges associated with the code from the programming side.

The development of plugins inside Inventor requires multidisciplinary knowledge. Firstly, one who is developing the program, should be familiar with the functionalities of Inventor, Secondly, knowledge of programming concepts like object-oriented programming, classes, interfaces, etc. should be understood since Inventor API is object-based, Thirdly to access those functionalities Inventor API should be deeply examined. Inventor API has extensive documentation on the Autodesk Inventor help website. However, it is not intuitive and requires a lot of trial and error before understanding it. Additionally, certain functionalities are not yet supported with API. Development of codes that automate assembly generation in Inventor although very useful and applicable in many cases when the structure is modular, is not supported with many tutorials or research. Most of the available sources concentrate on small-scale mechanical parts, and not covering complex geometries with many elements and parameters that influence one another, like in the case of greenhouses. These kinds of structures require the development of complex code that has to be thought true carefully. Creating this kind of code with Inventor is a narrow niche at the intersection of the disciplines and often finding resources and solutions for a problem is not easy.

One more difficulty is that iLogic does not have debugging systems like other environments for code development, like for example Visual Studio. It is possible to use Visual Studio for writing the code and copying it to iLogic, but not the other way around due to already uploaded libraries in iLogic, that should be manually added in Visual Studio. To debug in iLogic, MessageBox.Show and iLogicLog were used to follow the values of variables.

An important source of information during this process was the active community on Autodesk Inventor forum.

This page is intentionally left blank

6. CONCLUSION

In this master thesis, the primary objective was to explore the potential of automation of design phase in the context of greenhouse design, using Autodesk Inventor.

The study successfully demonstrated the application of parametric design principles using Inventor API and iLogic for greenhouse assembly generation. The developed framework and code provide a foundation for further research and development in automating other aspects of greenhouse design and even extend to different modular typologies. Automation of the design phase through parametrization and automated assembly generation can bring many benefits to companies whose main interest is modular structures developed inside of Inventor.

This master thesis has successfully explored and implemented parametrization in the context of greenhouse design using Autodesk Inventor. It has provided a practical solution and a customized tool to enhance efficiency and productivity in the design phase of greenhouse construction through the development of an external rule that automatically generates greenhouse assembly based on the user's inputs. The research findings contribute to the advancement of automation of the design phase of modular construction such as greenhouse structures.

While this research represents a significant step forward in automating greenhouse design, several additional topics could be addressed in future studies:

- To expand on this research, future studies could investigate ways to improve the modularity and scalability of the code, making it more adaptable to various greenhouse configurations. It would also be beneficial to explore alternative approaches to rectangular patterns. Furthermore, additional research should focus on implementing object-oriented programming in this case study, which has the potential to enhance modularity, simplify code complexity, facilitate easier code maintenance, and enable seamless integration of new features.
- Enriching the code with support for Gothic roof types and different frontal types would broaden its applicability to a wider range of greenhouse configurations.
- Investigating how to generate a model within a predefined plot and automate the connection between greenhouse structures and plots.
- Automation of Drawing Generation: Exploring methods to automate the generation of technical drawings and documentation for greenhouse structures.

REFERENCES

Agarwal, R., Chandrasekaran, S. and Sridhar, M. (2016) Imagining construction's digital future.

Ali Demir (2021) 'Modular Design, Design Standards, and Function Automation Using Inventor and iLogic'. Autodesk University.

Autodesk (no date) Getting Started with Inventor's API, Autodesk Inventor Professional 2024. Available at: https://help.autodesk.com/view/INVNTOR/2024/ENU/?guid=GUID-4939ABD1-A15E-473E-9376-D8208EC029EB (Accessed: 4 September 2023).

AWS (no date) What Is An API (Application Programming Interface)? [Online]. Available at: https://aws.amazon.com// (Accessed: 05 September 2023)

Barbosa Filipe et al. (2017) Reinventing construction: A route to higher productivity.

Changali, S., Mohammad, A. and Van Nieuwland, M. (2015) The construction productivity imperative.

Christian von Zabeltitz (2011) Integrated Greenhouse Systems for Mild Climates.

Vajna, S., Weber, C., Bley, H. and Zeman, K., 2009. CAx für Ingenieure: eine praxisbezogene Einführung. Springer-Verlag.

Von Elsner, B. et al. (2000) 'Review of structural and functional characteristics of greenhouses in European Union countries, part II: Typical designs', Journal of Agricultural and Engineering Research. Academic Press, pp. 111–126. Available at: https://doi.org/10.1006/jaer.1999.0512.

Von Elsner, B et al. (2000) REVIEW PAPER: Review of Structural and Functional Characteristics of Greenhouses in European Union Countries: Part I, Design Requirements, J. agric. Engng Res. Available at: http://www.idealibrary.com.

Ericsson, A. and Erixon, G. (1999) Controlling design variants: modular product platforms. Society of Manufacturing Engineers.

Estufasminho (no date) MANUAL DE MONTAGEM DE ESTUFAS.

Gembarski, P.C., Li, H. and Lachmayer, R. (2017) 'KBE-Modeling Techniques in Standard CAD-Systems: Case Study Autodesk Inventor Professional', in Managing Complexity – Proceedings of the 8th World Conference on Mass Customization, Personalization and Co-Creation (MCPC 2015). Berlin: Springer, pp. 215–233.

Girardet, A. and Boton, C. (2021) 'A parametric BIM approach to foster bridge project design and analysis', Automation in Construction, 126. Available at: https://doi.org/10.1016/j.autcon.2021.103679.

Hirz, M. et al. (2013) 'Knowledge-Based Design', in Integrated Computer-Aided Design in Automotive Development. Springer, Berlin, Heidelberg, pp. 309–330. Available at: https://doi.org/https://doi.org/10.1007/978-3-642-11940-8_1.

IBM (no date) What is automation? [Online]. Available at: https://www.ibm.com/topics/automation/ (Accessed: 05 September 2023)

Janssen, P. and Stouffs, R. (2015) 'Types of parametric modelling', in CAADRIA 2015 - 20th International Conference on Computer-Aided Architectural Design Research in Asia: Emerging Experiences in the Past, Present and Future of Digital Architecture. The Association for Computer-Aided Architectural Design Research in Asia (CAADRIA), pp. 157–166. Available at: https://doi.org/10.52842/conf.caadria.2015.157.

Jensen, P. et al. (2014) 'Developing products in product platforms in the AEC industry', in Computing in Civil and Building Engineering - Proceedings of the 2014 International Conference on Computing in Civil and Building Engineering. American Society of Civil Engineers (ASCE), pp. 1062–1069. Available at: https://doi.org/10.1061/9780784413616.132.

Jensen, P., Hamon, E. and Olofsson, T. (2009) PRODUCT DEVELOPMENT THROUGH LEAN DESIGN AND MODULARIZATION PRINCIPLES.

Jørgensen K. A. (2001) 'Product Configuration – Concepts and Methodology', in MANUFACTURING INFORMATION SYSTEMS. Aalborg: Department of Production Aalborg University, pp. 314–322.

Maurice van Sante (2022) Digitalisation must be top priority for construction companies [Online]. Available at: https://think.ing.com/ (Accessed: July 2023)

Monedero, J. (2000) Parametric design: a review and some experiences, Automation in Construction. Available at: www.elsevier.comrlocaterautcon.

Oxman, R. (2017) 'Thinking difference: Theories and models of parametric design thinking', Design Studies, 52, pp. 4–39. Available at: https://doi.org/10.1016/j.destud.2017.06.001.

Randy Shih (2014) Parametric Modeling with Autodesk Inventor 2014. SDC Publications.

Sandberg, M. et al. (2016) Design automation in construction-an overview.

Shah, J.J. (2001) 'Designing with Parametric CAD: Classification and comparison of construction techniques', in Geometric Modelling, pp. 53–68. Available at: https://doi.org/https://doi.org/10.1007/978-0-387-35490-3_4.

Sriram, D. et al. (1989) Knowledge-Based System Applications in Engineering Design: Research at MIT.

Stokes, M. (2001) Managing engineering knowledge: MOKA: methodology for knowledge based engineering applications.

Waguespack Curtis (2013) Mastering Autodesk Inventor 2014 and Autodesk Inventor LT 2014 . Wiley.

Woodbury, R. (Robert F. (2010) Elements of parametric design.

Wurzinger, A. (2016) Definitions to the Configuration Problem.

LIST OF ACRONYMS AND ABBREVIATIONS

API	Application Programming Interface
BIM	Building Information Modelling
CAD	Computer-Aided Design
CAE	Computer Aided Engineering
CAM	Computer Aided Manufacturing
CP	Configured Product
GUI	Graphical User Interface
IDF0	Icam Definition for Function Modeling
ICAM	Integrated Computer Aided Manufacturing
IT	Information Technology
KBE	Knowledge-Based Engineering
PP	Physical Product
PF	Product Family
UML	Unified Modeling Language