



Article

Advancing Sustainable Cyber-Physical System Development with a Digital Twins and Language Engineering Approach: Smart Greenhouse Applications

Ahmad F. Subahi

Department of Computer Science, University College of Al Jamoum, Umm Al-Qura University, Makkah 21421, Saudi Arabia; afsubahi@uqu.edu.sa

Abstract: In recent years, the integration of Internet of Things technologies in smart agriculture has become critical for sustainability and efficiency, to the extent that recent improvements have transformed greenhouse farming. This study investigated the complexity of IoT architecture in smart greenhouses by introducing a greenhouse language family (GreenH) that comprises three domain-specific languages designed to address various tasks in this domain. The purpose of this research was to streamline the creation, simulation, and monitoring of digital twins, an essential tool for optimizing greenhouse operations. A three-stage methodology was employed to develop the GreenH DSLs, a detailed metamodel for enhanced smart monitoring systems. Our approach used high-level metamodels and extended Backus–Naur form notation to define the DSL syntax and semantics. Through a comprehensive evaluation strategy and a selected language usability metrics, the expressiveness, consistency, readability, correctness, and scalability of the DSL were affirmed, and areas for usability improvement were highlighted. The findings suggest that GreenH languages hold significant potential for advancing digital twin modeling in smart agriculture. Future work should be aimed at refining usability and extending its application range. The anticipated integration with additional model-driven engineering and code generation tools will improve interoperability and contribute to digital transformation in the smart greenhouse domain and promote more sustainable food production systems.

Keywords: digital twins; domain specific languages; smart greenhouses; model-driven engineering; internet-of-things; language family



Citation: Subahi, A.F. Advancing Sustainable Cyber-Physical System Development with a Digital Twins and Language Engineering Approach: Smart Greenhouse Applications. *Technologies* **2024**, *12*, 147. <https://doi.org/10.3390/technologies12090147>

Academic Editor: Spyridon Nikolaidis

Received: 1 July 2024

Revised: 20 August 2024

Accepted: 26 August 2024

Published: 2 September 2024



Copyright: © 2024 by the author. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

The application of emerging technologies has been gaining increasing attention in industry and academia. Interest has increased exponentially in some manufacturing and process-related contexts, leading to changes in industries and transforming the way businesses operate. These changes have various benefits, such as optimizing electronic service delivery and predicting potential system failures [1–3].

Digital twins are a cutting-edge technology that have revolutionized various industries by virtualizing and mirroring every aspect of a physical real-world object and simulating its operational processes. The virtual object represents the physical object in the digital space and provides specialists with meaningful feedback from the virtual world. The feedback is derived from the historical and current behavior of objects and can be used to optimize business performance [2,4].

By leveraging the power of the Internet of Things (IoT), artificial intelligence, data analytics, and machine learning with model-driven and engineering (MDE) and languages engineering (LE), digital twins establish a dynamic connection between the physical and the digital realms, and create a wealth of opportunities and potential use cases in various domains. According to recent predictions by several world-leading technological research

and consulting organizations, such as Gartner and Global Market Insight, the market size of this technology is expected to expand in the next 10 years because many executives across a wide spectrum of sectors plan to incorporate digital twins into their businesses [1,5].

One study [6] discusses how IoT technology could be used to improve the efficiency and sustainability of smart greenhouse farming. It emphasizes the importance of sensor interfaces for monitoring and controlling environmental elements such as temperature, humidity, and soil moisture, thereby enabling an automated farming ecosystem. By combining IoT with traditional agricultural practices, crop productivity and resource management can be improved, thereby laying the groundwork for the development of enhanced sensor interfaces in agricultural environments.

In addition, the transformational effect of smart sensors and data analytics on precision agriculture has been investigated [7]. This review discusses the evolution of sensor technology from basic data collection tools to sophisticated systems that are capable of complex data processing and real-time decision-making. It concludes that the integration of smart sensors not only enhances the granularity and precision of agricultural data but propels a shift toward more autonomous and sustainable agricultural practices that align with the broader goals of modern agricultural innovation.

Furthermore, MDE is an advanced software development technique that uses models as the primary artifact for handling complexity and change in software systems. It enables developers to concentrate on the fundamental structure and behavior of a system by, first, specifying its features using high-level models, before converting these models into executable code. MDE accelerates development cycles and enables organizations to adapt quickly to market changes and technical improvements [8]. It is considered a digital transformation tool because it enables organizations to adjust their systems quickly to accommodate new processes and technologies while causing minimal disruption [9].

1.1. Brief History of Digital Twin Technology

Digital twin technology originated in the 1970s from simulated environments or mirrored systems that were created in the aerospace domain by NASA. These systems were developed to monitor spacecraft during missions, such as Apollo 13, whose oxygen tanks exploded a few days after launch. By using a mirrored system environment, the NASA flight control team at the time were able to model, test, and simulate possible solutions more quickly and economically and with less negative consequences than would be possible in real life. Today, this approach is known as digital twin technology [2,3].

The concept of digital twins can be broken down into three key components: real space, virtual space, and digital thread [10]. Real space represents physical, real-world objects, such as people, processes, entities, and phenomena. Virtual space is used to represent the real-world objects virtually. The digital thread is used to link these two spaces: data flow from the real space to the virtual space, and vice versa.

Furthermore, digital twins can be classified into three categories based upon level of integration, whether digital model, digital shadow, or digital twin. In digital models, data are exchanged manually between physical and digital entities because the transformation in the state of the physical object is not reflected in the digital state and vice versa. In turn, in digital shadows, automatic data exchange is supported in one direction: the changes in the physical object can be reflected in the digital one. In turn, in digital twins, automatic bidirectional data exchange between physical and digital objects is supported [11].

A significant advancement of digital twin development is the ChatTwin platform. It introduces a segment and generates a workflow that enhances the generation of scene descriptions, thereby reducing errors and improving efficiency. This platform automates the document generation process by using techniques such as integer programming to boost the accuracy and quality of digital twin documents [12]. Other research presented a new Large Languages Model (LLM-powered) multi-agent system that combines the qualities of digital twins and multi-agent architectures to provide a cognitive understanding of

physical systems from local to global views and, notably, in the time dimension [13]. These intelligent agents demonstrate good security and usability in provided scenarios.

1.2. *Advances in the Agriculture Sector*

Agriculture is widely recognized as a pivotal sector that plays a key role in the global economy. It safeguards human existence by ensuring sustainability of food production over the long term. Agricultural sustainability can be achieved by increasing the application of emerging technologies in the field of farming. This investment approach has the potential to enhance food production, thereby improving the extent to which the demands of a growing population are met [14,15].

Sensors are core components in IoT-based architecture and play critical roles in the agricultural domain [16–18]. Sensors collect information on various environmental parameters that affect crop production, such as soil nutrients, temperature, and humidity, and pass it on to embedded control systems. According to [19], an IoT device (FarmFox) was designed to collect, analyze, and monitor real-time soil information, including chemical and biological properties, such as pH, potassium, and nitrate levels. The analyzed data are transmitted to farmers and provide information on the health of the soil and associated recommendations to increase productivity. FarmFox and other types of sensors are connected via wireless sensor networks, which are a widely recognized and used communications technology in smart agriculture solutions; other technologies are ad hoc networks and cognitive radio [19,20]. Wireless sensor networks have unique features that make them appropriate for use in smart agriculture, such as the capability to self-establish, self-configure, and self-recover, and the ability to operate with other devices, such as radio-frequency-based sensors and microcontrollers for controlling smart farming automation and conducting environmental monitoring [20].

By combining digital twin technology with smart agriculture, further advances may be achieved in this field. Digital twin technology can virtualize crop production, greenhouse structures and shapes, internal greenhouse atmosphere, and other physical entities and processes. This technology could be used to dynamically reflect data relating to temperature and humidity control, soil and irrigation, and CO₂ and light monitoring that are collected from smart greenhouses over time, and across different physical systems and crop management systems [21].

1.3. *Model-Driven Engineering as a Transformative Approach in Smart Agriculture*

MDE emerged to address increasing systems complexity and the multiplicity of systems implementation platforms. This problem resulted from increased levels of abstraction during the development process, which arose from using models as class artifacts instead of writing code. According to MDE practitioners [22], stakeholders can benefit from MDE in various ways, such as improved productivity and quality or reduced costs. MDE also supports various activities in the software development life cycle. Thus, MDE can contribute to the sustainability of software development.

As discussed in [23,24], combining MDE with digital twin technology may bring about increased benefits, particularly if combined further with language engineering via domain-specific languages (DSLs). In the context of this research, modeling digital twins of a special case of cyber-physical systems or smart greenhouse monitoring systems can be achieved. Such combinations of technologies can significantly advance the smart agriculture sector. This approach enables experts in the field, who may not necessarily possess modeling skills, to benefit from emerging technologies such as digital twins. It will enable them to build their desired system by providing supporting tools that have creation, simulation, and analysis capabilities. The proposed approach consists of the following components:

1. An abstracted software system architecture for a digital twin IoT monitoring system that demonstrates the concepts and components of the whole system.
2. A DSL family, or GreenH, for modeling and designing a digital twins-based greenhouse monitoring system. GreenH comprises three DSLs, with each one being respon-

sible for capturing a specific view of the domain: GreenH Design DSL for representing the structural view including IoT smart devices, crop growth, and greenhouse characteristics; GreenH Flow DSL for capturing greenhouse behavior in terms of data flow and control, including data pipelines and data transmitting format and controlling conditions; and GreenH Twin for representing the structural and behavioral views of the smart greenhouses in the virtual environment, thereby forming the corresponding system.

3. Formal definitions of the abstract and concrete syntax and semantics of the proposed languages, using a metamodeling approach and the extended Backus–Naur form (EBNF) notation, respectively.
4. A model transformation engine to deduce GreenH Twin elements from the GreenH Design and GreenH Flow models.
5. An execution engine/model transformation and code generation strategy to support the automation of creating digital twin simulation systems for monitoring smart greenhouses.

Figure 1 illustrates the key components of the proposed approach. It is worth mentioning that this paper focuses on the design of the GreenH family of DSLs, which support the creation of digital twins to model smart greenhouse systems and monitor their performance. The other components of the proposed approach are discussed briefly. A detailed description of the code generation strategy and the supporting tool are beyond the scope of this paper.

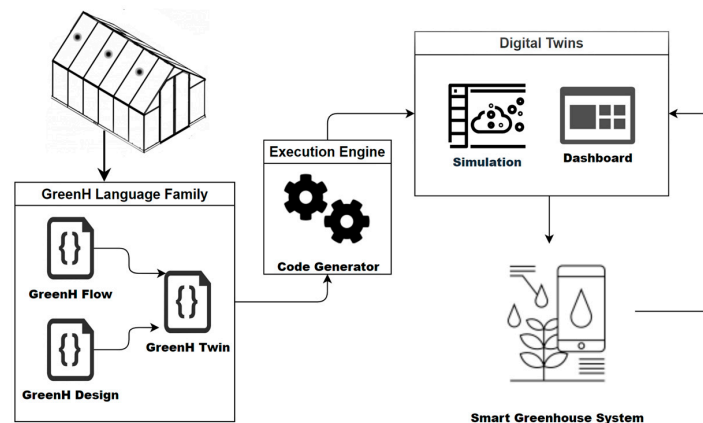


Figure 1. Components of the proposed approach.

The contribution of this paper is a novel MDE approach with an accompanying proof-of-concept GreenH DSL collection, alongside its syntax and constructs. These DSLs capture an abstract view of IoT-based monitoring systems for greenhouses and, specifically, support modeling digital twins to simulate the process performance of smart greenhouses under different conditions. The DSL approach used in this study may have several advantages over LLMs and generative approaches. According to other studies that utilized LLMs and generative models for digital twins development [12,13,25], the following points demonstrate the advantages of our DSL-based approach over other approaches:

- The specificity of DSLs enables quicker iterations and modifications, which are essential in dynamic IoT environments, unlike LLMs, which may require extensive tuning or many user prompts.
- DSLs focus on specific functionalities, use available resources more efficiently, and avoid the generality overhead associated with LLMs and generative approaches.
- DSLs adhere to specific standards within their domain and promote interoperability among systems, which poses challenges for LLMs across different domains.

Even though we focused mainly on this proof-of-concept version of two main components of IoT ecosystems of greenhouses sensors and actuators, other key components are

also considered in real-world settings to form IoT architecture for smart greenhouses, as illustrated in Figure 2.

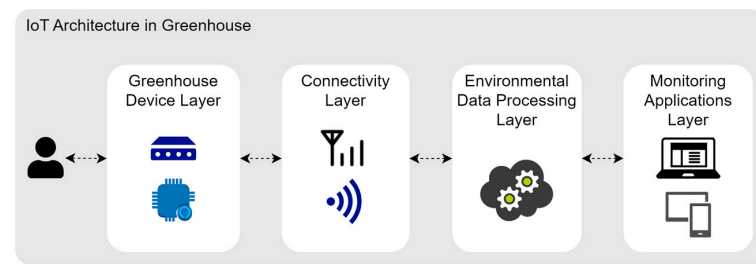


Figure 2. Components of IoT ecosystem in the domain of smart greenhouses.

After providing an overview of related work in Section 2, we present a three-stage methodology for designing the promising DSL of this study. The GreenH language family is discussed in Section 3. The design principles of the syntax and semantics of GreenH Design, GreenH Flow, and GreenH Twin are presented in Section 4 as a high-level metamodel and EBNF notation. Section 5 describes and formalize the concrete syntax of GreenH DSLs using EBNF notation. Section 6 presents a case study of a running example of a simplified version of a greenhouse environment. The example consists of an IoT temperature monitoring system that motivates the use of the proposed DSL in modeling digital twins and reflecting the physical assets of the system. In addition, we discuss the development of an evaluation strategy for the language to measure its performance in a constructive manner. The evaluation strategy adopted in this work measures the usability of DSL with regard to a defined criterion. A detailed discussion and analysis of the evaluation results are given in Section 7. Section 8 provides a conclusion, summarizes the work, and provides suggestions for future research.

2. Related Research

This section groups related research into three categories: the parameters used in smart IoT-based systems, the use of digital twins in the agriculture domain, and the application of MDE and language engineering approaches in simulation systems development. As discussed in [14], traditional greenhouses can benefit from IoT technologies to become smart by adapting and using monitoring, measurement, and control systems in the crop production environment. These types of systems can collect data on temperature, humidity, and other parameters via wireless networks to monitor greenhouses comprehensively. Furthermore, IoT system control and monitoring can be achieved via several techniques, such as fuzzy logic, proportional–Integral–derivative controllers, and artificial neural networks.

Table 1 summarizes the common IoT systems considered by related research. It is clear that the majority of the proposed smart systems consider parameters, such as temperature, humidity, light intensity, and soil moisture to be critical crop growth or cultivation parameters.

Table 1. Summary of common related IoT systems.

Ref.	Parameters	IoT Systems
[15]	Temperature, humidity, light intensity, pH value, and CO ₂ level	Temperature, light control, air pollution, and soil moisture monitoring
[16]	Temperature and energy	Temperature control (PID)
[18]	Temperature and energy	Temperature control (Fuzzy logic)
[26]	temperature and humidity	Temperature control, humidity monitoring
[27]	Temperature, humidity, luminosity, and CO ₂	Temperature control, humidity, luminosity, and CO ₂ monitoring
[28]	Soil moisture, humidity, and temperature	Watering management system
[29]	CO ₂ levels, light intensity, and humidity	Soil irrigation system and more

2.1. Digital Twins in Smart Agriculture

2.1.1. Digital Twins in Internet of Things

The authors of [30] present a prototype of a digital twin for smart farming that is based on IoT technology. The technology is used to control an irrigation system that refers to decisions made jointly by farmers and artificial intelligence. Several smart IoT systems associated with farms, such as weather stations or irrigation systems, are connected to the cloud via a special gateway that sends information to an IoT infrastructure. The collected information is analyzed in the cloud and visualized using a number of dashboards, and input into the real-world system through the cloud or by connecting controllers to the IoT system.

A digital twins conceptual framework for designing and implementing farm management systems is introduced and discussed by [31]. This framework defines the concept of digital twins and introduces a typology that is based on the life cycle phases of the objects being virtualized. Five common smart farming use cases, taken from the European IoF 2020 project, are used in the application and validation of the framework: arable farming, dairy farming, greenhouse horticulture, organic vegetable farming, and livestock farming.

A low-cost digital twin framework for smart agriculture was developed to tackle the issue of farmers' hesitancy to use smart systems because of the associated hardware and maintenance costs [31]. AgriLoRa creates digital twins that enable farmers to watch and monitor soil parameters and the status of their farms in near real time. The framework consists of cloud servers that run two different computer vision algorithms: MobileNet and UNet. These are connected via wireless sensor networks and are used to detect plant diseases and other related nutrient deficiencies [32].

2.1.2. Digital Twins in Controlled Environment Agriculture

Another interesting application of digital twins in controlled environment agriculture (CEA) systems is presented by [33]. The main objective of CEA is to improve the efficiency of production, to optimize plant yields, and to enhance product quality by controlling plant growth and the surrounding atmosphere. The authors of [33] propose a platform-independent digital twin architecture for optimizing productivity, with a balance between production and resource consumption. A simulation model considers different crop microclimate controls in the architecture to manage the productivity of the agriculture system. The proposed architecture was applied to a greenhouse prototype for validation purposes.

Furthermore, in ref. [34], a monitoring digital twin was designed to monitor different subsystems of CEA facilities. This architecture can collect and store different types of CEA data, extract and assess relevant information, and then visualize the results. A case study discusses a glass greenhouse with specific dimensions that was used as a CEA lab. A test bench was used to test two samples of plants that require different fertigation. Smart devices, including sensors and actuators, were allocated in a simple way. They adopted the 4C model (connect, collect, control, change) to operate physical objects. In addition, digital twin requirements defined by [35] were considered in the requirements elicitation phase, which includes services, extensibility, interaction, data storage, and data integration.

2.2. Languages Engineering Approach in Modeling and Simulation Systems

Over the past decade, DSLs have attracted a great deal of interest in the field of software engineering. One application is in the development of simulation systems. DSLs are designed to express solutions concisely and effectively within a specific problem domain, thereby enabling domain experts to concentrate on the essential aspects of the problem without being hampered by the syntax and constructs of general-purpose programming languages. The use of DSLs in simulation systems enables the creation of efficient simulation models that can be used for automatic code generation.

The development of simulation systems using DSLs has numerous benefits, such as increased overall productivity, including enhanced code readability, maintainability, and reusability. By providing a high-level declarative syntax that closely corresponds to domain

terminology and concepts, DSLs enable a more natural expression of simulation models and logic, as well as overall system characteristics and components. This results in a more intuitive comprehension of system behaviors. Table 2 summarizes some of the DSLs that are used for simulation.

Table 2. Summary of related DSL utilized in different simulation systems.

Ref.	DSL	Purpose	Language	Approach	Type
[36]	SESSL	Writing simulation experiments	Scala	LE	Embedded
[37]	ScalaTion	Writing clear, concise, and intuitive simulation programs	Scala	LE	Embedded
[38]	RobotML	Designing, simulating, and deploying robotic applications	DSL	MDE	Metamodel
[39]	SimulateIoT	Designing simulation environments for IoT systems	DSL	MDE	Metamodel
[40]	SimulateIoT-FIWARE	Designing simulation environments for IoT systems for FIWARE platform.	DSL	MDE	Metamodel
[41]	DEVS-Ruby	Modelling and simulation of discrete event system specification (DEVS)	DSL	MDE	Finite StateAutomata

2.3. Domain-Specific Languages and the Agriculture Sector

Recently, there has been massive investment in emerging technologies in the agricultural sector, and radical transformation has enhanced productivity and sustainability in different ways. It has optimized planting and harvesting, reduced waste, and improved crop yields. With these advances in the field, agriculture faces key challenges to achieving its long-term sustainability and economic goals and values. These challenges are associated with the design and implementation of large-scale systems to meet stakeholder needs. Like other technologies, DSLs have, in different forms, occasionally, aided in the development process of agriculture systems to improve productivity, reduce the gap, and develop high-quality systems [42].

A publication by [42] introduces a DSL framework for developing precision agriculture systems. Domain-driven design methodology is utilized for formalizing the domain analysis process, which is based on the feature modeling technique. A number of features are developed to visually represent the shared and variant features of the domain of precision agriculture. The framework adopts a collection of DSLs, each one of which is used for modeling a specific aspect of the domain. The findings of a controlled experiment show that the proposed approach demonstrates good potential to enhance collaboration between domain experts and DSL developers.

Critical concerns for developing a knowledge-based service system in the agricultural domain (Cyber-Brain) are presented and discussed by [43]. These concerns are enhancing non-specialist users' access to large, unstructured, and scattered agriculture-related information, and adequately providing connected information to knowledge workers. This knowledge service is provided by two encompassing approaches to collecting agricultural-related knowledge from multiple sources: knowledge engineering and language engineering. A domain-specific ontology is designed to be utilized for aggregating knowledge from various sources, including domain experts, digital archives, and the Web, in practical scenarios. A domain-specific ontology must be developed because current ontologies are too generic to be adopted for particular applications. Moreover, there is a challenging issue regarding how to keep the existing ontology up to date [44].

3. Methodology

In this section, the steps taken to develop the GreenH language family are outlined and discussed. The methodology can be described as a systematic three-phase approach to developing a DSL (Figure 3). The main phases are (1) domain analysis, (2) defining DSL requirements, and (3) designing and implementing the DSLs. In the first phase, relevant

domain knowledge of smart greenhouses was gathered through a literature review and analysis of existing domain solutions. This process was used to identify key domain concepts, stakeholders, and main operations. Moreover, the main components of digital twin architecture and its internal design, which should be considered in designing the DSLs, are also included in the domain analysis phase.

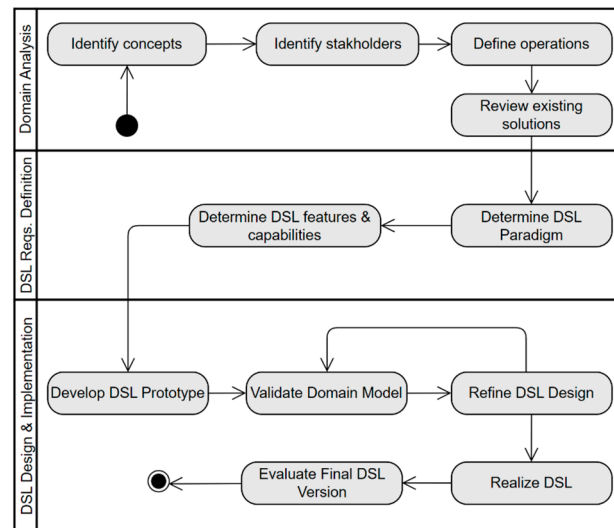


Figure 3. The proposed three-phase approach for developing DSL.

To reduce complexity, the domain analysis phase is executed informally in the majority of formal DSL development methodologies. However, ontologies can be applied, formally, at the early stages of a domain analysis phase to conceptualize domain knowledge with sufficient expressive power for domain concepts, properties, and relationships. The advantages of ontology significantly enhance DSL development by reducing or preventing the occurrence of errors, providing reasoning and querying, and defining language semantics [45].

However, even though using ontologies holds benefits for DSL development, the approach requires an extra transformation step, from ontology to DSL grammar. In addition to knowing DSL requirements, this requires a steep learning curve to understand what the ontology describes and how it was designed to build the transformation engine. Moreover, adequate knowledge of special technologies such as RDF (Resource Description Framework) triplets, OWL (Web Ontology Language) for representing facts, and SPARQL (SPARQL Protocol and RDF Query Language) for querying RDF semantic triple stores is required [46]. When ontologies grow in complexity, performance considerations increase the complexity of data manipulation, which requires more sophisticated queries and operations, as well as optimization techniques for querying and updating large volumes of triples [45].

The primary motivation for choosing a DSL is its ability to provide a tailored solution for the expression of concepts and operations in terms that are directly aligned with domain-specific logic and workflows. The development process can be designed using a minimal number of transformation steps. For instance, parser generation, with its comprehensive toolset for grammar handling, simplifies and accelerates complex and time-consuming parts of DSL development.

In the second phase, the language requirements are defined. This phase includes the selection of the DSLs type, determining whether the language is external or internal (embedded), and whether it has textual, graphical, or hybrid notations. The rationale behind these decisions is considered. In the third phase, the initial prototyping of the DSLs is described by emphasizing the creation of basic language constructs, syntax design, and semantics. A loop for collecting stakeholder feedback is applied to improve the design of

the DSLs until the final version of the language is achieved. Next, the final version of the DSLs is implemented using appropriate tools, frameworks, and technologies. After that, the DSLs are evaluated to ensure that they accurately represent the domain and achieve their intended purpose.

3.1. Domain Analysis

The foundational goal of our approach is to produce a DSL that will simplify the development of digital twins for IoT-based greenhouse climate monitoring systems. Therefore, a domain analysis of relevant systems was conducted to understand the complex details and identify the key elements of smart greenhouse operations, the relationships between these concepts, and the role of IoT technologies in greenhouse environment monitoring. The following subsections summarize the key information identified in this phase to be considered in the language development approach, which included the design choices of the DSL and its syntax and constructs.

3.1.1. Temperate Fruits Development and Growth

Temperate fruits such as apples, strawberries, and cherries experience a complex growth process that is influenced by environmental factors. These fruits require distinct seasonal changes, including winter dormancy with sufficient chilling hours for breaking bud dormancy and synchronizing flowering and fruit set. The development of these progresses through stages that include bud formation, flowering, fruit set, growth, and maturation, all regulated by external factors such as temperature, light, and water (see Table 3). Achieving optimal fruit development necessitates a precise balance of these conditions, with adequate winter chilling that is followed by favorable growing season temperatures and light. Understanding these requirements is crucial for optimizing production and maintaining high-quality fruit across various climates [47].

Table 3. Development stages of some temperate plants.

Stage No.	Apple	Strawberry
1	Purpose: Dormancy and Bud Break; Temp: below 7 °C; Period: 3–4 months	Purpose: Planting and Root Development; Temp: 18 Day–12 Night °C; Period: 1–2 months
2	Purpose: Growth of Leaf and Floral Initiation; Temp: 21–24 °C; Period: 1–2 months	Purpose: Vegetative Growth; Tempe: 25 Day–12 Night °C; Period: 2–3 months
3	Purpose: Fruit Development and Growth; Temp: 22 Day–12 Night °C; Period: 30–40 days	Purpose: Flowering and Fruiting; Temp: 25 Day–12 Night °C; Period: 3–4 months
4	Purpose: Late Fruit Maturation and Harvest; Temp: 22 Day–12 Night °C; Period: 3–4 months	-

3.1.2. Types of Greenhouses

The performance of a greenhouse, which is typically measured by elements such as heat preservation, light transmission and distribution, and energy efficiency, can be greatly affected by the shape of the greenhouse.

Depending on the environment, the kind of plants being produced, and the particular needs of the greenhouse operation, different greenhouse design forms have different benefits and drawbacks [48,49]. It is not the goal of this study to identify the optimal greenhouse design with the best performance [50], nor to identify the best internal climate conditions for growing certain types of crops [51]; instead, our intention is to determine the ideal distribution and number of sensors inside the greenhouse, and to develop an appropriate placement strategy.

3.1.3. Sensors: Types, Number and Locations

During the domain analysis, we identified several types of IoT systems that can be tracked and monitored via different parameters. Table 4 indicates monitoring parameters that can be considered in language design.

Table 4. Summary of considered parameters and their associated IoT systems.

IoT System	Parameter to Measure	Measurement Unit
Temperature change parameter	Internal heating temperature	Celsius
Humidity change parameter	Internal humidity level	Percentage
Light change parameter	The intensity of light on surface	Lux
Energy consumption parameter	Electrical power consumed by the system.	Watt
Communication parameter	Data transmission between IoT components	Bits per second

Determining the appropriate number of sensors in a greenhouse is important because distributing smart sensors in random numbers and positions may increase the complexity of a wireless network unnecessarily; it may also cause significant errors in profile measurement inside a greenhouse [52]. After the domain analysis is completed, a simple technique is developed to select the appropriate number of sensor nodes based on general principles of several mathematical and engineering domains. A set theory and formal calculus definition is utilized to describe the parameters and develop the following equation:

$$N = \frac{A}{C} * K * H$$

where:

- N is the number of sensors. Each sensor is located in a single zone for microclimate controlling and monitoring tasks.
- A is the measurable floor area of the greenhouse.
- C is a constant to represent the nominal coverage area for a sensor. Then, a coverage function $c : \mathbb{R} \rightarrow \{0, 1\}$ can be defined where:

$$\begin{cases} 1, & \text{within the sensor range} \\ 0, & \text{out of the sensor range} \end{cases}$$

- Let H be the height of the greenhouse; for some shapes, we consider the height in the middle, where:

$$\begin{cases} 3 \text{ m or less}, & H = 1 \\ \text{More than 3 m}, & H = +0.05 \text{ for each additional meter} \end{cases}$$

Let K be the shape coefficient constant that adjusts for the complexity of the greenhouse's shape and internal structures. If $\mathcal{S}$ is the set of all possible shapes (σ) with the same floor area ($\sigma \in \mathcal{S}$), then K can be seen as: $K = K(\sigma)$, where:

$$\begin{cases} \text{Simple shape}, & K = 1.0 \\ \text{Multi span}, & K = 1.2 \\ \text{Curved}, & K = 0.9 \\ \text{Quonset}, & K = 1.1 \\ \text{Gable roof}, & K = 1.2 \\ \text{Gothic arch}, & K = 1.2 \end{cases}$$

Selecting optimal positions for installing a limited number of sensors enables accurate monitoring of the internal environment inside of a greenhouse [53]. According to [54], the accuracy of microclimate data depends strongly on the number and placement of sensors. Therefore, optimizing sensor quantity and locations is essential for precise microclimate measurement and effective cultivation system control.

3.1.4. Simulation and Visualization

The domain analysis also examined opportunities for adopting digital twins to simulate the performance of smart greenhouses. Doing so gave rise to an understanding of the different digital twin architectures that can be used to express and model smart IoT systems via our initiative, GreenH DSL [55,56]. Based on this analysis, the high-level architectural design choice of the digital twins was determined, which consisted of two main components: (1) a simulation subsystem and (2) a dashboarding subsystem, as discussed in the introduction (see Figure 1).

3.2. Definition of Requirements of Domain Specific Languages

The main aim of GreenH is to guide greenhouse specialists in modeling, in an expressive way, the structure, behaviors, and interactions of various digital twin IoT monitoring system components inside greenhouses, thereby enabling them to create applications to monitor the performance and energy consumption of a system. This section briefly discusses the desired features of GreenH by defining its high-level requirements from three perspectives: the expressive scope, the level of abstraction, and the language constructs. For instance, when considering the case of a greenhouse that needs to maintain optimal plant growth conditions via adjusting the system behavior at a specific development stage, the GreenH Flow language can express the detail of the targeted humidity level in respect of these conditions.

The expressive scope is concerned with the ability of the developer to use the language to express the main properties of the IoT greenhouse monitoring system. In the example mentioned, the language must express the controlling behaviors and constraints of the system adequately. Regarding the targeted abstraction level of the language, we focus on minimizing the complexity of the language syntax far as possible. It can be noted from the example (Listing 1) that the controlling process and action have straightforward and declarative names, with clear and organized block structures; each has start ({} and end ({}). The controlling rule follows a simple ‘when then’ pattern, which simplifies understanding and editing conditions and actions.

Listing 1. A snapshot of GreenH Flow to show a humidity monitoring and control process.

```
HumidityControl {
  Type: "Threshold"
  Stage: "@1"
  GreenhouseControlRule { when: "humidity > 70" then: "IncreaseVentilation" }
}
IncreaseVentilation {
  Operation: "Increase"
  Duration: "temperature <= 25 and humidity <= 68"
}
```

Furthermore, the language constructs are concerned with the ability of the language to describe the syntax and semantics required to represent all key concepts of greenhouse components associated with the controlling purpose. Using keywords such as *GreenhouseControlRule*, *when*, and *then* provides a clear, structured syntax that ensures that the DSL is both readable and easy to write. In addition, the semantics of the DSL are highlighted through the condition action pairings of each rule. These pairs demonstrate the ability of the language to describe behavior based on specific environmental readings.

4. Design Principles of the GreenH Language Syntax

The abstract syntax of the GreenH DSL collection is specified by using metamodels, which define their key concepts and the main relationships between them (Figure 4). Other studies [57–59] have found that this commonly used approach demonstrates adequacy and efficiency in defining the abstract syntax of various DSLs. Figure 4a–c shows that the core concepts of GreenH DSLs are organized into Unified Modeling Language (UML) classes

based on their functionality and their role in the language in order to demonstrate the interrelationships between the language components at a high level.

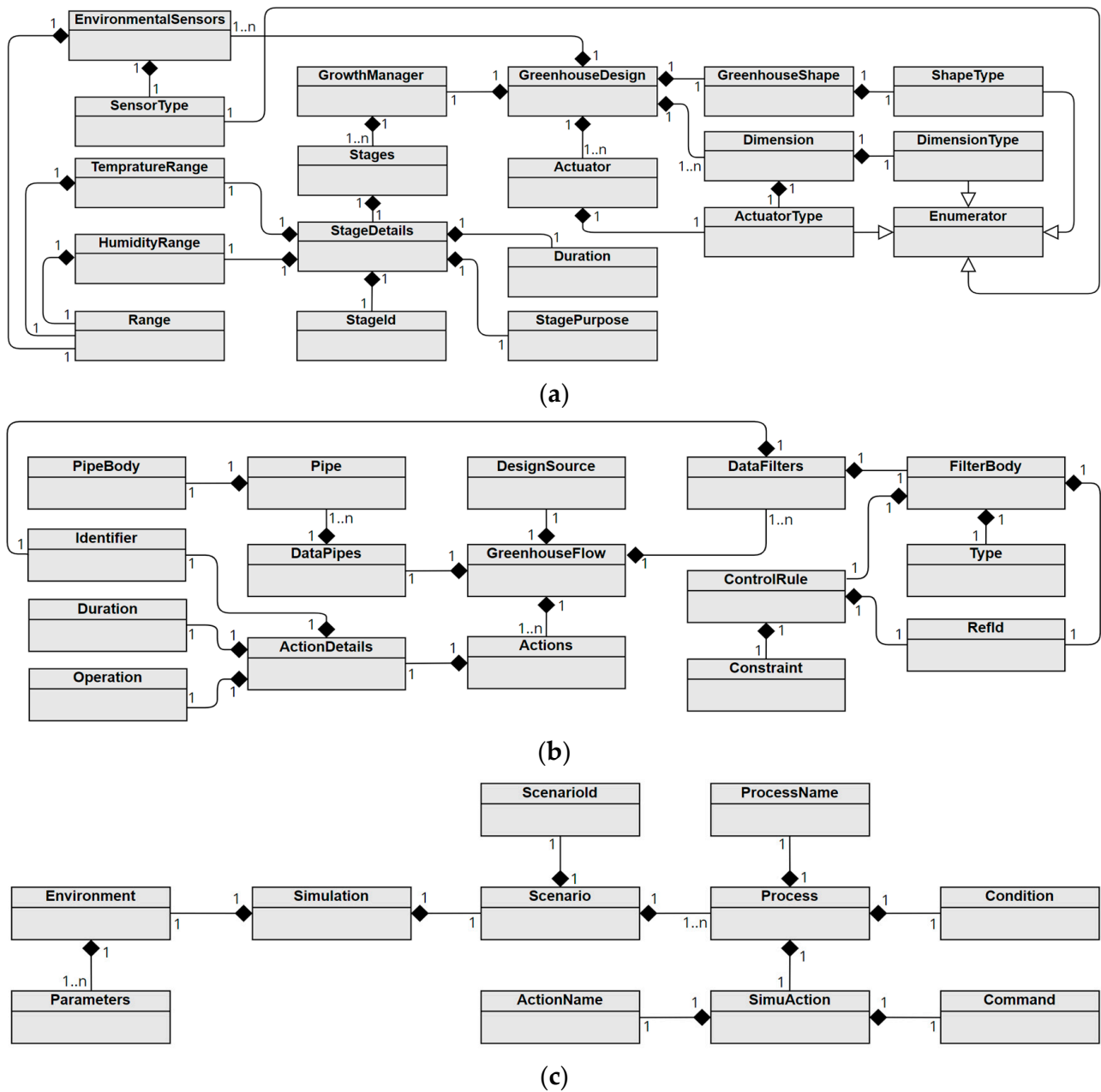


Figure 4. (a) Metamodel of the GreenH Design DSL. (b) Metamodel of the GreenH Flow DSL. (c) Metamodel of the GreenH Twin DSL.

4.1. Aspect Orientation and Separation of Concerns

Our design choice involved developing a separate DSL metamodel to represent each aspect of the language. GreenH Design focuses on the physical and structural greenhouse components, while GreenH Flow manages logical data flows and operational control and constraints and so on. Combining these into a single metamodel may increase its complexity. Furthermore, the strategy of considering separate metamodels ensures high cohesion within these metamodels and low coupling between them. It also produces better design modularity and maintainability of the metamodels.

By applying our design choice, we created the following metamodels:

- (1) The metamodel of GreenH Design DSL, which encapsulates essential greenhouse characteristics: shape, dimensions, and growth development stages (Figure 4a). Moreover, it expresses its key IoT system components: environmental sensors, actuators, and conditions, and their interrelations.
- (2) The metamodel of GreenH Flow DSL, which encapsulates critical control processes and constraints in terms of data flow and data filters (Figure 4b), thereby supporting diverse greenhouse monitoring and control cases.
- (3) The metamodel of GreenH Twin, which encapsulates the virtual entities and behavior corresponding to both physical and logical concepts that appear in the previous metamodels, thereby forming various digital twin simulation scenarios (Figure 4c).

4.2. Integration Functionality of GreenH Language

Note that the integration between the two DSLs is not complex. GreenH Design sets up the greenhouse context and components, which are likely foundational for GreenH Flow DSL. There is an inherent dependency, because the smart devices, defined in the first DSL, are directly utilized in the data flow specifications of the second one.

This seamless integration feature highlights the adaptability and coherence of GreenH and ensures that the design and operational aspects of the greenhouse are coordinated well. The following example demonstrates how the structural definitions in GreenH Design are directly leveraged by GreenH Flow to automate responses based on real-time environmental data, thereby ensuring that optimal conditions are maintained with minimal manual intervention. Once the GreenH Design defines a sensor layout, for instance a temperature sensor in the greenhouse, GreenH Flow can readily access and incorporate the information provided by the sensor into its control logic. Doing so minimizes redundancy and promotes efficient data usage across the system, as illustrated by Listings 2 and 3.

Listing 2. A snapshot of GreenH Design to show an example of specifying a configuration.

```
Greenhouse Design {
  Shape (type="Dome")
  Dimensions { length: 50m; width: 20m; height: 15m; }
  Environmental Sensors { TemperatureSensor(id="T1", location="center", range="18-22°C") }
}
```

Listing 3. A snapshot of a corresponding GreenH Flow to show the predefined context that creates a smart control rule.

```
GreenhouseControlRule {
  when "TemperatureSensor(id='T1', location='center') exceeds 22°C"
  then "Activate(CoolingSystem(id='C1', zones='center'))"
}
```

4.3. Extendibility Features of GreenH

Distributing the GreenH architecture across separate metamodels that represent different aspects of a greenhouse monitoring system enhances its extendibility. This separation allows developers to extend or modify parts of the DSL without it affecting other unrelated components. For instance, consider the demand to incorporate advanced environmental monitoring capabilities to predict conditions that could lead to plant diseases. To address this demand, a new type of sensor that measures not only humidity and temperature but also soil pH and electrical conductivity is added to the language. Extending GreenH to accommodate this demand requires minimal modifications: (1) extending the sensor definition in the GreenH Design DSL metamodel, and (2) updating the controlling rule in the GreenH Flow DSL, as demonstrated in Listings 4 and 5.

Listing 4. A snapshot of GreenH Design EBNF definition to extend the language with new features.

```

<SensorType> ::= "TemperatureSensor" | ... | "MultiPurposeSensor"
<SensorProperty> ::= <TemperatureRange> | ... | <PHRange> | <ConductivityRange>
<PHRange> ::= "acidic" | "neutral" | "alkaline"
<ConductivityRange> ::= "low" | "medium" | "high"

```

Listing 5. A snapshot of GreenH Flow to show the controlling rule corresponds to the extension.

```

GreenhouseControlRule {
  when "MultiPurposeSensor(id='S3', location='westField', pH='acidic', conductivity='high')"
  then "Adjust(SoilTreatment(id='ST1', action='limeApplication', zones='westField'))"
}

```

4.4. Testability Features of the Language Design

The testability of GreenH is supported by its clear and structured design and syntax. Its expressive constructs facilitate the formal verification process. Each language component of the DSL is defined by using distinct and modular syntax rules, which can be tested individually for correctness. For example, the syntax for defining sensors and actuators ensures that each element is declared with the necessary identifiers and attributes, which allows for automatic validation of their presence and format. This modularity also enables precise unit testing and integration testing of each component.

Additionally, the explicit declaration of conditions and actions the GreenhouseControlRule is using facilitates formal testing techniques. Tests can be written to specifically verify that the system behaves as expected under various environmental scenarios determined by the rules. To demonstrate this, model checking, as a formal verification technique, can be used to ensure that the language behaves consistently and as intended across all different scenarios. This technique systematically explores all possible states of a system to verify whether certain properties hold. In this work, a symbolic model verifier based on the NuSMV tool (Version 2.6.0) [60] was developed as proof of concept for the testability feature of GreenH, and is presented in Section 7.4.

5. Definition of GreenH Language Syntax

The context-free grammar of EBNF notation is used to describe the concrete syntax of GreenH. This is an approach that is commonly used to define the syntax of its language family, as indicated in [58,59]. The developed DSLs were then used to implement the system scenario discussed in the example we provide (Section 6). This step was necessary to revise and validate a real-world system, and to arrive at the final version of the language.

Moreover, the use of context free grammar for formalizing the concrete syntax and considering greenhouse-related terminologies in naming the DSL concepts ensured that the language is both precise and user-friendly, thereby enabling users to easily specify and modify system behavior according to different conditions. This dual focus on adaptability and usability directly satisfies the core requirements of efficiency and flexibility in greenhouse monitoring.

5.1. Formalizing GreenH Concrete Syntax

The corresponding concrete syntax of GreenH DSLs is defined using EBNF rules. EBNF is a commonly used notation technique for and approach to defining the formal specification language; it uses context-free grammar to define the concrete syntax and structure of DSLs in a formal mathematical way [61,62]. The "::<=" syntax of EBNF notation is used to define a rule, "{}" indicates repetition of zero or more instances, "[]" indicates an optional element, "|" refers a choice (logical OR), and "<>" denotes non-terminal symbols.

The developed DSLs were used to implement the system scenario discussed in Section 6. This step is necessary to revise and validate a real-world system and to arrive at the final version of the language.

5.1.1. Formalizing the Concrete Syntax of GreenH Design DSL

The GreenH Design language was designed to express the structural and physical system components of greenhouses. The following definitions of design principles (core language concepts) constitute the metamodel (Figure 4a):

- *Greenhouse Design*: a concept of the language that expresses the syntax and constructs of the overall structure of a greenhouse design, and that encompasses its shape, dimensions, sensors, actuators, and growth journey details as enclosed in a specific format. This can be formally defined using EBNF notation as:

```
<GreenhouseDesign> ::= "Greenhouse Design {" <GreenhouseShape> <Dimensions> <EnvironmentalSensors> <Actuators> <GrowthManager> "}"
```

- *Greenhouse Shape*: a concept of the language that specifies the geometric shape of the greenhouse, which is a critical factor in determining the required number of sensors and their positions inside the greenhouse area. The common types of greenhouse shapes, such as square, rectangle, and dome, are captured using *ShapeType* to provide more flexibility in design options. This can be formally defined using EBNF notation as:

```
<GreenhouseShape> ::= "Shape (type=" <ShapeType> ")"
<ShapeType> ::= "Square" | "Rectangle" | "Dome" | "Quonset-shaped" |
"GableRoof-shaped" | "GothicArch-shaped"
```

- *Dimensions*: a concept of the language that describes the syntax and constructs the dimensions of the greenhouse, which are fundamental for planning the size and area along with its shape. This can be formally defined using EBNF notation as:

```
<Dimensions> ::= "Dimensions {" <DimensionList> "}"
<DimensionList> ::= <Dimension> {<Dimension>}
<Dimension> ::= <DimensionType> ":" <Value> ";"
<DimensionType> ::= "length" | "width" | "height" | "radius"
```

- *Environmental Sensors*: a concept of the language for describing the syntax and constructs of special types of sensors, installed in specific locations, and their tasks, which are utilized for monitoring environmental conditions inside the greenhouse, such as humidity, temperature, and soil moisture. Initial readings can be determined to enable monitoring and control of the smart system according to an acceptable range. This can be formally defined using EBNF notation as:

```
<EnvironmentalSensors> ::= "Environmental Sensors {" <SensorList> "}"
<SensorList> ::= <EnvironmentalSensor> {<EnvironmentalSensor>}
<EnvironmentalSensor> ::= <SensorType> "(id=" <Literal> ",
location=" <Literal> <OptionalRange> ")"
<SensorType> ::= "SoilMoistureSensor" | "TemperatureSensor"
<OptionalRange> ::= ["", range=" <Range> "]" | ""
```

- *Actuator*: a concept of the language that expresses the syntax and constructs of actuators that can act on various greenhouse components that are placed in specific locations. It can carry out actions in response to rules for actively managing the environment inside the greenhouse. The actuator can be specialized into different types of actuators for specific greenhouse systems, such as heating and lighting systems using the *ActuatorType* element. This can be formally defined using EBNF notation as:

```
<Actuators> ::= "Actuators {" <ActuatorList> "}"
<ActuatorList> ::= <Actuator> {<Actuator>}
<Actuator> ::= <ActuatorType> "(id=" <Literal> ", zones=" <Literal> ")"
<ActuatorType> ::= "IrrigationSystem" | "CoolingSystem" | "HeatingSystem"
| "lightingSystem"
```

- *Growth Manager*: a concept of the language for expressing the growth and development stages of a specific type of crop planted inside the greenhouse. As explained in Section 3.1.1, the temperature required by temperate fruits vary for each stage of growth development. The details of each growth stage, including its purpose, duration, and optimal environmental conditions, are captured in the language using the *Stage* element. This enables the language that describes the growth journey in detail, which can be defined formally using EBNF notation as:

```

<GrowthManager> ::= "GrowthManager { " "crop: " <Literal> "; " <StageList> " }"
<StageList> ::= <Stage> { <Stage> }
<Stage> ::= "Stage { " <StageDetails> " }"
<StageDetails> ::= <StageId> <StagePurpose> <Duration> <TemperatureRange>
<HumidityRange>
<StageId> ::= "stage Id: " <Number> "; "
<StagePurpose> ::= "stage Purpose: " <Literal> "; "
<Duration> ::= "duration: " <Value> "; "
<TemperatureRange> ::= "temperature Range: { " <Range> " }"; "
<HumidityRange> ::= "timidity Range: { " <Range> " }"; "

```

5.1.2. Formalizing the Concrete Syntax of GreenH Flow DSL

The GreenH Flow language is specifically designed to express the complex management and automation of environmental control systems in greenhouses. Its primary purpose is to enable precise configuration and control of data flows, filtering, and automated responses within these controlled environments. By defining data pipelines, filters, controllers, and actions in an organized and comprehensible format, GreenH Flow allows users to tailor the conditions within a greenhouse to optimize plant growth and resource efficiency. This language helps in seamlessly integrating sensors, actuators, and control systems, making it possible to maintain ideal conditions for various plant species by monitoring and reacting to real-time environmental data. The following design principles definitions (core language concepts) constitute the metamodel (Figure 4b):

- *Greenhouse flow*: a concept of the flow language used to express syntax and construct the overall data flow behavior of the greenhouse smart system. It is worth mentioning that the greenhouse components that are defined previously using GreenH Design language can be referenced in this model via *design_source* element. These can be formally defined using EBNF notation as:

```

<GreenhouseFlow> ::= "Greenhouse Flow { " <DesignSource> <DataPipes>
<DataFilters> <Actions> " }"
<DesignSource> ::= "design_source: " "{ " <Literal> " }"

```

- *Data Pipes*: a concept of the flow language used to express syntax and construct the data flow within the system, defining how data are taken in from sensors, processed, and output. It contains one or more *Pipe* elements, each specifying a single data pipeline within the system, including the input source of data, the filter applied, and the directed control system. This can be formally defined using EBNF notation as:

```

<DataPipes> ::= "DataPipes { " { <Pipe> } " }"
<Pipe> ::= "Pipe(id= \" \" <Identifier> \" \") { " <PipeBody> " }"
<PipeBody> ::= "system: \" \" <Identifier> \" \" \"input: \" \" <identifiers> \" \" \"filter: \" \" <identifier> \" \" \"output: \" \" <identifier> \" \" \"

```

- *Data Filters*: a concept of the flow language used to express the syntax and constructs of filters used to process data within pipes. It comprises of multiple definitions of two types of elements, namely *DataFilter* and *ControlRule*. Each filter has a type (e.g., threshold, range) and an associated growth development stage. On the other hand, the rule element describes the condition(s) that dictate how data are filtered and action triggering based on data conditions. This can be formally defined using EBNF notation as:

```

<DataFilters> ::= "DataFilters { " { <DataFilter> } }"
<DataFilter> ::= <Identifier> "{" <FilterBody> "}"
<FilterBody> ::= "type: \" \" <Type> \" \" \"stage: \" \" <RefId> \" \" \"
<ControlRule> \" \"
<ControlRule> ::= "ControlRule { \" \"when: \" \" <Constraint> \" \"
\"then: \" \" <RefId> \" \" }"

```

- *Actions*: a concept of the flow language used to describe the syntax and constructs of the collection of actions. Each Action describes a specific operation to be performed by the system in a particular duration based on data conditions and decision-making logic, expressed in a relevant *DataFilter* element. This can be formally defined using EBNF notation as:

```

<Actions> ::= "Actions { \" { <ActionDetails> } }"
<ActionDetails> ::= <Identifier> "{" "operation: \" \" <Operation> \" \"
\"duration: \" \" <Duration> \" \" }"

```

5.1.3. Formalizing the Concrete Syntax of GreenH Twin DSL

The GreenH Twin language is specifically designed to express the digital twin simulation-based system corresponding to a physical IoT-based monitoring system in greenhouses. This language model serves as a basis for the later automatic code generation phase, aiming at producing simulation programs (code) that simulate a real-world IoT system for a target platform (e.g., Python or Java). The following core language concepts constitute the metamodel (Figure 4c):

- *Virtual World*: an element of the digital twins language used to bind the virtual representations of all greenhouse structural and behavioral components corresponding to real-world ones expressed in GreenH Design and GreenH Flow DSLs. This can be formally defined using EBNF notation as:

```

<VirtualWorld> ::= "Virtual World { \" { <WorldBody> } }"
<WorldBody> ::= "{" <SensorInstance> <Simulation> "}"

```

- *Sensor Instance*: an element of the digital twins language used to represent a virtual entity from a corresponding one that is expressed in GreenH Design language. This can be formally defined using EBNF notation as:

```

<SensorInstance> ::= "sensor_instance { \" { <InstanceDetails> } }"
<InstanceDetails> ::= "id: \" <Identifier> \"type: \" \" <Type> \" \"
\"position: \" \" <Position> \" \" "

```

- *Simulation*: an element of the digital twins language used to create simulations for monitoring/testing scenarios. It provides the capability to monitor the system behavior in various real-world scenarios based on real data. The *Simulation* element comprises the simulation environment settings and scenarios. This can be formally defined using EBNF notation as:

```

<Simulation> ::= "Simulation { \" <Environment> <Scenario> }"
<Environment> ::= "Environment { \" <Parameters> }"
<Parameters> ::= <Parameter> | <Parameter> <Parameters>
<Parameter> ::= "parameter: \" <Identifier> \"= \" <Value>

```

- *Agricultural Scenario*: a concept of the language that acts as a contextual container to group a series of processes that relate to the simulation. It can be utilized to define a specific set of circumstances under which the contained processes will operate. Each process includes a name, a condition under which the process should be executed, and an action to be performed if the condition is met. This can be formally defined using EBNF notation as:

```

<Scenario> ::= "Scenario" <ScenarioId> "{" { <Process> } }"
<ScenarioId> ::= "(id= \" <Literal> \")"

```

```

<Process> ::= "Process {" <ProcessName> <Condition> <SimuAction> "}"
<ProcessName> ::= "name:" <Literal>
<Condition> ::= "condition:" <Condition>

```

- *Action*: a concept of the language that describes the action to trigger when the condition of a process is met. The details include the name and the command to execute. This can be formally defined using EBNF notation as:

```

<SimuAction> ::= "Simulation Action:" "{" <SimuActionDetails> "}"
<SimuActionDetails> ::= <ActionName> <Command>
<SimuActionName> ::= "name:" <Literal>
<Command> ::= "command:" <Identifier> "=" <Value>
<Duration> ::= "Duration:" <Duration>

```

6. Use Case: Digital Twin Internet of Things Temperature Monitoring System

In this section, we introduce a digital twin IoT temperature monitoring system that was designed specifically for greenhouse environments. This solution mirrors the live status, working conditions, and position of its real-world counterpart through sensors and data analytics. Consider a simple rectangular greenhouse, 30 m in length, 20 m wide, and 4 m high, in a temperate climate where temperature control is critical to cultivate temperature-sensitive plants such as apples. Traditional monitoring methods rely on periodic manual readings and reactive adjustments, which can be labor-intensive and imprecise. The system we propose integrates a network of temperature sensors distributed throughout the greenhouse that continuously collect data on micro-climatic conditions. These data feed into the digital twin model, which is hosted on a cloud-based platform that is accessible to the greenhouse operators. The following subsections demonstrate the different aspects of the greenhouse using GreenH DSLs.

Representing a Simple Monitoring System Using GeenH DSLs

We use an example to illustrate the implementation of a simple digital twin model for a greenhouse temperature monitoring system. The structural specifications, including the shape and dimensions of the greenhouse, are expressed. In addition, the list of all temperature sensors and the cooling system are modeled as greenhouse sensors and actuators using GreenH Design language (Listing 6). This listing also represents the required temperature range at each growth development step via the greenhouse growth manager element.

In addition, a data flow is set up to monitor temperature readings and trigger events when the temperature is out of range. There are connections between components and a representation of system behavior and responses. The following GreenH Flow snapshot, expressed in Listing 7, demonstrates how the control rule activates an action to operate the cooling system for 60 min through the data filter element when the temperature within the greenhouse moves beyond the desired range. The filter is applied to the first stage of apple growth development described in GreenH Design.

At this point in the language development process, GreenH demonstrates its ability to express a real-world scenario using two GreenH DSLs in a declarative way. It is important to mention that we are aware of the rest of the development process, which involves producing intermediate representations of the language or even executable code generation directly. In our case, the development process requires translating both the GreenH Design and Flow language models into the GreenH Twin model to act as an intermediate representation of the digital twin simulation system. Note that the design of the execution engine or model translator responsible for performing this process and the final code generation step were beyond the scope of this work. However, the mapping of elements of the source models can be seen in the following snapshot of the translated GreenH Twin model (Listing 8).

Listing 6. A snapshot of GreenH Design to represent the structure of the greenhouse.

```

GreenH Design {
  Shape (type="Rectangle")
  Dimensions { length: 30m; width: 20m; height: 4m; }
  Sensors {
    TemperatureSensor(id="TS1", location="Zone 1", range="15 C to 35 C")
    TemperatureSensor(id="TS2", location="Zone 2", range="15 C to 35 C ")
    TemperatureSensor(id="TS2", location="Zone 3", range="15 C to 35 C ")
  }
  Actuators {
    CoolingSystem(id="HeatSys1", zones="Zone 1, Zone 2, Zone 3")
  }
}
GrowthManager {
  Crop: "Apple"
  Stage {
    StageId: 1
    StagePurpose: "Dormancy and Bud Break"
    Duration: 105 day
    TemperatureRange: [21 C to 26 C]
  }
}
}

```

Listing 7. A snapshot of GreenH Flow to represent the data flow and control in the greenhouse.

```

GreenH Flow {
  DataPipes {
    Pipe(id="TempratureDataPipe") {
      Input: "TS1, TS2"
      Filter: "TempratureRangeFilter"
      Output: "TempratureController"
    }
  }
  DataFilters {
    TempratureRangeFilter {
      Type: "Range"
      Stage: "@1"
      ControlRule {when: "temprature > 26", then: "IncreaseCooling"}
    }
  }
  Actions {
    IncreaseCooling {
      Operation: "Increase"
      Duration: "60 min"
    }
  }
}

```

Listing 8. A snapshot of the translated GreenH Twin representing the corresponding digital twin simulation system within the greenhouse.

```

Virtual World {
  Virtual Instances {
    TempSensor1 { id: "@TS1", from: GreenH.Desgin}
    TempSensor2 { id: "@TS2", from: GreenH.Desgin}
  }
  Simulation {
    Environment {
      Parameter: "TimeUnit" = "minutes"
      Parameter: "MaxTime" = "500"
      Parameter: "Temprature" = "0"
    }
    Scenario(id="Temperature Cooling") {
      Process {
        Name: "Morning Check"
        Condition: "TempSensor1 > 26 C or TempSensor2 > 26 C"
        Simulation Action: {
          Name: "IncreaseCooling"
          Command: "IncreaseCooling = start"
          Duration: "60 min"
        }
      }
    }
  }
}
}

```

7. Language Evaluation Strategy

To obtain the full benefit of GreenH, it is important that its languages meet certain usability and satisfaction criteria associated with their intended purpose and the target user experience. Conforming to these criteria will enable target users to utilize the developed DSL more easily and more effectively [63]. Therefore, we used the example presented in Section 6 to evaluate the GreenH DSLs in a systematic and constructive manner. An evaluation strategy was developed, focusing on the opinions of peer experts in the field, to determine whether they would be able to use these DSLs to model a real-world digital twin monitoring system for greenhouses.

7.1. Expert Participant Characteristics

The proposed GreenH DSLs cover different domains: software and language engineering, IoT system development, and agriculture. A comprehensive evaluation thus requires multiple experts with different backgrounds related to the DSL. To achieve this, we considered seven PhD-level experts from related fields who each had more than four years of experience in academia; three were experts in software engineering, two were experts in IoT-based system development, two were agriculture experts, and one was an expert in botany.

According to Jakob Nielsen's principle of diminishing returns, which was developed for heuristic evaluation [64], the benefit gained from each additional evaluator tends to decrease beyond a certain point, typically after the fifth evaluator. This principle is grounded in empirical research that demonstrates that the first few evaluators are likely to identify the majority of usability issues, with subsequent evaluators uncovering progressively fewer new problems. Therefore, adding more evaluators beyond this number tends to yield minimal additional insights relative to the increased effort required, thereby illustrating the principle of diminishing returns.

Applying Nielsen's principle of diminishing returns [64] to the usability evaluation of GreenH DSLs is beneficial because of the diverse backgrounds of the experts assessing the relevant DSL domains. This approach ensures an efficient and comprehensive assessment of DSL usability with respect to each associated domain, and that the most critical usability concerns are identified effectively.

7.2. Evaluation Strategy and Criteria

The strategy is concerned with the quality of the language structure and its ability to model differently sized system models. Six key criteria are considered in the evaluation: expressiveness, usability, consistency, readability, correctness, and scalability (Table 4). The evaluation approach proposed by this research was inspired by various existing usability evaluation frameworks for DSL, such as those proposed by [63,65,66]. These works were discussed previously and justify different quality characteristics that are relevant to the success of DSL utilization during the development process. The UML activity diagram in Figure 5 demonstrates the overall evaluation process conducted by experts for all languages.

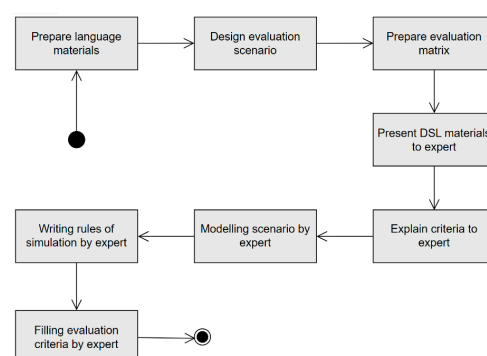


Figure 5. Detailed process of the GreenH DSLs evaluation strategy.

As would be the case for evaluation frameworks, a number of quality characteristics of the languages were adopted (Table 5). Several criteria were excluded from the evaluation process, such as extendibility, integration, performance, testability, tools support, error handling, maintainability, and documentation. These criteria are more relevant for evaluating DSL performance after completing the implementation stage.

Table 5. Key criteria considered in the evaluation process.

ID	Criteria	Description
C1	Expressiveness	Ability to model all necessary domain concepts accurately and completely.
C2	Readability	Clarity of the language syntax.
C3	Usability	Ease of use and learning for new users.
C4	Consistency	Uniformity in language constructs and their usage.
C5	Correctness	Enforcement of domain constraints to avoid invalid models.
C6	Scalability	Ability to handle different sizes and complexities of models.

7.3. Results Discussion and Final Remarks

One round of evaluation was conducted, and it involved two focus group meetings to present and evaluate all GreenH languages. The first meeting was attended by four experts with backgrounds relating to software systems, and the second meeting involved three experts with agriculture and botany backgrounds. Participants' opinions on each criterion were aggregated to indicate the overall strengths of the DSLs and potential areas for improvement. Table 6 summarizes the performance of the GreenH DSLs according to the average scores obtained in each category.

Table 6. Summary of GreenH IoT DSL performance.

ID	Score (1)	Score (2)	Score (3)	Score (4)	Score (5)	Score (6)	Score (7)	Average Score	Summary of Expert Notes
C1	4	4	4	3	3.5	4	3	3.64	Missing constructs for advanced scenarios in GreenH Design, such as network connection protocols and other types of smart devices.
C2	4	5	4	4	4	3	3	3.86	DSLs syntax is clear for target users in GreenH Design and Flow DSLs; further simplification is recommended in the Twin DSL.
C3	4	3	4	3	4	3	2.5	3.00	Difficulties might appear with more complex simulation scenarios. Enabling the execution engine to generate more boilerplate code is required or consideration of a template-based approach for code generation.
C4	5	5	4	4	5	4	3	4.29	DSLs constructs are consistent and each DSL focuses on a particular aspect of the language.
C5	4	4	4	3.5	3	3	3	3.50	The language is defined and restricted to a metamodel and EBNF. Further correctness tests are required in later stages of development.
C6	5	3.5	4	4	4	3.5	4	4	The DSLs can be scalable; advanced experiments are required

The results of the evaluation indicate that the GreenH DSLs are consistent and readable and scale effectively, a strong foundation for correctness. However, the scores also reveal that to achieve the full potential of the language, GreenH Design could be improved significantly in the area of expressiveness and usability. While the score for expressiveness suggests that GreenH is capable of capturing the majority of domain-specific constructs using concepts related to smart greenhouses, it could be enhanced by adding further constructs to include more critical domain concepts or refine existing constructs for greater precision.

Furthermore, the score for language usability indicates that while the language is usable, users face notable obstacles. These include the effort required to learn to use the language and the absence of supportive tooling at this stage of the development process. Additionally, poor technical and programming skills of participants may affect usability. Given that the final implementation and its associated editors are not complete, it is vital to

address these challenges through research on the user experience of using the language with its associated editor.

Regarding language readability, consistency, and correctness, the score indicates that GreenH DSLs are relatively well designed, with clear syntax and semantics that make the language constructs understandable. Its level of consistency reflects the degree to which the language design follows uniform principles, which ensures predictability and easy learning. Moreover, the language correctness score indicates that the DSLs behave correctly in given scenarios. Comprehensive testing, including formal method techniques, may be required, where applicable, to provide formal evidence that the language consistently adheres to its specified behavior.

The scalability score shows that the language can be adapted and scaled well to perform efficiently as the extent of usage grows. However, the language may encounter challenges with very large projects that involve highly complex simulation scenarios expressed in GreenH Twin. All insights gained from this round of evaluation will guide the next iterations of DSL development. By considering user feedback, employing user-centered design principles, and rigorously testing the language, we plan to enhance its overall effectiveness and accelerate broader adoption in the target domain illustrated in Figure 2.

7.4. GreenH Verification Using Model Checking

This section illustrates a formal approach, as a proof of concept, to verifying GreenH Flow DSL for greenhouse management using the model checking tool NuSMV [58,60]. We focus on a simplified segment of the language that was designed for managing temperature within a greenhouse. The aim of this section is to provide proof of concept for the testability of the proposed language, not to provide a full verification of the language. Figure 6 shows the verification steps conducted in this work.

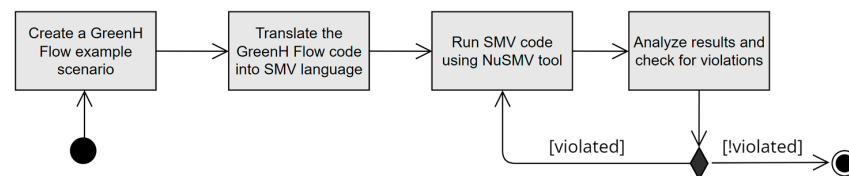


Figure 6. Model checking process for verifying the language using NuSMV tool.

The considered GreenH Flow DSL example involves components for data piping, filtering, and executing actions based on specific temperature conditions. This can be seen in Listing 9.

Listing 9: The GreenH Flow DSL example used for verification.

```

ControlRule {
    when: "temperature > 26", then: "IncreaseCooling"
    when: "temperature <= 22", then: "DecreaseCooling"
}

...
Actions {
    IncreaseCooling {
        Operation: "Increase"
        Duration: "60 min"
        Level: "High" # Added detail about the level of increase
    }
    DecreaseCooling {
        Operation: "Decrease"
        Duration: "30 min"
        Level: "Low" # Added detail about the level of decrease
    }
}

```

Next, we translated the DSL code into SMV code, which involves defining state variables and specifying the behavior of the system through logical constraints. Two types of key properties are defined:

1. Safety property:
 - When the temperature is above 26 degrees and the cooling level is not already high, the system transitions the cooling level to high.
 - When the temperature falls to 22 degrees and lower and the cooling level is not already low, the system transitions the cooling level to low.
 - Cooling system actions are gradually phased out by decrementing the timer and turning off the system when the timer reaches 1.
2. Liveness property:
 - When the temperature exceeds 26 degrees, the cooling system will eventually be set to high.
 - When the temperature falls to 22 degrees or lower, the cooling system will eventually be set to low.

After running the model checker with the NuSMV tool and verifying all properties, the output confirmed that the specifications are true, indicating that our example behaves correctly under the defined conditions. This can be seen in Figure 7.

```
*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://minisat.se/MiniSat.html
*** Copyright (c) 2003-2006, Niklas Een, Niklas Sorensson
*** Copyright (c) 2007-2010, Niklas Sorensson

-- specification AG (temperature > 26 -> AF cooling_level = high) is true
-- specification AG (temperature <= 22 -> AF cooling_level = low) is true
```

Figure 7. The output results shows the verification of the system behavior.

8. Conclusions

In this work, we introduce the GreenH language family, which is a new DSL developed through a novel MDE approach, that aims to enhance IoT architecture in smart greenhouses and advance digital twin modeling. Our declarative design streamlines the description, simulation, and monitoring of IoT systems to facilitate the construction of digital twins, which are critical for optimizing greenhouse operations. We have presented a detailed three-stage methodology for crafting three GreenH languages, namely Design, Flow and Twin DSL, alongside metamodeling architectures for smart monitoring systems based on digital twins. The principles underpinning the syntax and semantics of the GreenH DSLs were articulated using high-level metamodels and EBNF notation.

A robust evaluation strategy for all GreenH DSLs was developed and analyzed, and the findings support its expressiveness, consistency, readability, correctness, and scalability. To guide future enhancement, opportunities for improving usability were identified. Our future efforts will be informed by feedback and evaluations and will concentrate on refining the usability of the GreenH languages and widening their scope for different IoT applications and scenarios in agriculture. We also foresee further integration with other MDE and code generation tools to expand their interoperability. The NuSMV model checking tool was applied for further verification of the language to verify the behavior of the control system modelled by the language.

In future work, we aim to refine the GreenH DSLs by enhancing their syntax and semantics, and to achieve improved robustness and greater user-friendliness by utilizing the ANTLR (Another Tool for Language Recognition) parser generator for efficient data processing performance. Essential support tools, such as visual editors and performance analyzers, will also be developed to facilitate GreenH use in real-world applications. Additionally, a sophisticated code generator will be implemented to convert the language scripts into executable code for simulating smart greenhouse operations. These advancements

will form a comprehensive transformation chain that enhances the development life cycle of digital twin applications, thereby significantly accelerating digital transformation in smart agriculture.

We believe that our research makes a significant contribution to advancing and accelerating the digital transformation of smart agriculture and the greenhouse field. GreenH could be a valuable asset for the IoT system design community, particularly for creating and refining smart greenhouse monitoring systems using digital twins technology. Its ongoing development promises to fuel research into smart agriculture, thereby fostering more sustainable and more efficient food production practices.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in the study are included in the article, further inquiries can be directed to the corresponding author.

Conflicts of Interest: The author declares no conflicts of interest.

References

1. Attaran, M.; Celik, B.G. Digital Twin: Benefits, use cases, challenges, and opportunities. *Decis. Anal. J.* **2023**, *6*, 100165. [CrossRef]
2. Anshari, M.; Hamdan, M. Enhancing e-government with a digital twin for innovation management. *J. Sci. Technol. Policy Manag.* **2023**, *14*, 1055–1065. [CrossRef]
3. Armeni, P.; Polat, I.; De Rossi, L.M.; Diaferia, L.; Meregalli, S.; Gatti, A. Digital twins in healthcare: Is it the beginning of a new era of evidence-based medicine? A critical review. *J. Pers. Med.* **2022**, *12*, 1255. [CrossRef]
4. Singh, M.; Fuenmayor, E.; Hinchy, E.P.; Qiao, Y.; Murray, N.; Devine, D. Digital Twin: Origin to Future. *Appl. Syst. Innov.* **2021**, *4*, 36. [CrossRef]
5. Costello, K.; Omale, G. *Gartner Survey Reveals Digital Twins Are Entering Mainstream Use*; Gartner Inc.: Stamford, CT, USA, 2019; Available online: <https://www.gartner.com/en/newsroom/press-releases/2019-02-20-gartner-survey-reveals-digital-twins-are-entering-mai> (accessed on 22 February 2024).
6. Rayhana, R.; Xiao, G.; Liu, Z. Internet of things empowered smart greenhouse farming. *IEEE J. Radio Freq. Identif.* **2020**, *4*, 195–211. [CrossRef]
7. Soussi, A.; Zero, E.; Sacile, R.; Trincherio, D.; Fossa, M. Smart Sensors and Smart Data for Precision Agriculture: A Review. *Sensors* **2024**, *24*, 2647. [CrossRef]
8. Bucchiarone, A.; Cabot, J.; Paige, R.F.; Pierantonio, A. Grand challenges in model-driven engineering: An analysis of the state of the research. *Softw. Syst. Model.* **2020**, *19*, 5–13. [CrossRef]
9. Kamburjan, E.; Sieve, R.; Prabhu, C.; Amato, M.; Barmina, G.; Occhipinti, E.; Johnsen, E.B. GreenhouseDT: An Exemplar for Digital Twins. In Proceedings of the SEAMS 2024, Lisbon, Portugal, 15–16 April 2024.
10. Barricelli, B.R.; Casiraghi, E.; Fogli, D. A survey on digital twin: Definitions, characteristics, applications, and design implications. *IEEE Access* **2019**, *7*, 167653–167671. [CrossRef]
11. Li, M.; Wang, R.; Zhou, X.; Zhu, Z.; Wen, Y.; Tan, R. ChatTwin: Toward automated digital twin generation for data center via large language models. In Proceedings of the 10th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation, Istanbul, Turkey, 15–16 November 2023; pp. 208–211.
12. Sun, Y.; Zhang, Q.; Bao, J.; Lu, Y.; Liu, S. Empowering digital twins with large language models for global temporal feature learning. *J. Manuf. Syst.* **2024**, *74*, 83–99. [CrossRef]
13. Björnsson, B.; Borrebaeck, C.; Elander, N.; Gasslander, T.; Gawel, D.R.; Gustafsson, M.; Jörnsten, R.; Lee, E.J.; Li, X.; Lilja, S.; et al. Digital twins to personalize medicine. *Genome Med.* **2020**, *12*, 4. [CrossRef]
14. Bersani, C.; Ruggiero, C.; Sacile, R.; Soussi, A.; Zero, E. Internet of Things Approaches for Monitoring and Control of Smart Greenhouses in Industry 4.0. *Energies* **2022**, *15*, 3834. [CrossRef]
15. Tripathy, P.K.; Tripathy, A.K.; Agarwal, A.; Mohanty, S.P. MyGreen: An IoT-enabled smart greenhouse for sustainable agriculture. *IEEE Consum. Electron. Mag.* **2021**, *10*, 57–62. [CrossRef]
16. Subahi, A.F.; Bouazza, K.E. An intelligent IoT-based system design for controlling and monitoring greenhouse temperature. *IEEE Access* **2020**, *8*, 125488–125500. [CrossRef]
17. Arora, N.K. Agricultural sustainability and food security. *Environ. Sustain.* **2018**, *1*, 217–219. [CrossRef]
18. Fei, X.; Xiao-Long, W.; Yong, X. Development of energy saving and rapid temperature control technology for intelligent greenhouses. *IEEE Access* **2021**, *9*, 29677–29685. [CrossRef]
19. Sengupta, A.; Debnath, B.; Das, A.; De, D. FarmFox: A quad-sensor-based IoT box for precision agriculture. *IEEE Consum. Electron. Mag.* **2021**, *10*, 63–68. [CrossRef]

20. Pandey, C.; Sethy, P.K.; Behera, S.K.; Vishwakarma, J.; Tande, V. Smart agriculture: Technological advancements on agriculture—A systematic review. *Deep. Learn. Sustain. Agric.* **2022**, *1*, 1–56.
21. Verbruggen, C.; Snoeck, M. Practitioners' experiences with model-driven engineering: A meta-review. *Softw. Syst. Model.* **2023**, *22*, 111–130. [\[CrossRef\]](#)
22. Peterson, T.A. Systems engineering: Transforming digital transformation. In Proceedings of the INCOSE International Symposium, Orlando, FL, USA, 20–25 July 2019.
23. Govindasamy, H.S.; Jayaraman, R.; Taspinar, B.; Lehner, D.; Wimmer, M. Air quality management: An exemplar for model-driven digital twin engineering. In Proceedings of the 2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), Fukuoka, Japan, 10–15 October 2021.
24. Bordeleau, F.; Combemale, B.; Eramo, R.; Van Den Brand, M.; Wimmer, M. Towards model-driven digital twin engineering: Current opportunities and future challenges. In Proceedings of the Systems Modelling and Management: First International Conference, ICSMM 2020, Bergen, Norway, 25–26 June 2020.
25. Palchunov, D.; Vaganova, A. Methods for Developing Digital Twins of Roles Based on Semantic Domain-Specific Languages. In Proceedings of the 2021 IEEE 22nd International Conference of Young Professionals in Electron Devices and Materials (EDM), Souzga, Russia, 30 June–4 July 2021; pp. 515–519.
26. Hernández-Morales, C.A.; Luna-Rivera, J.M.; Perez-Jimenez, R. Design and deployment of a practical IoT-based monitoring system for protected cultivations. *Comput. Commun.* **2022**, *186*, 51–64. [\[CrossRef\]](#)
27. Yang, J.; Liu, M.; Lu, J.; Miao, Y.; Hossain, M.A.; Alhamid, M.F. Botanical internet of things: Toward smart indoor farming by connecting people, plant, data and clouds. *Mob. Netw. Appl.* **2018**, *23*, 188–202. [\[CrossRef\]](#)
28. Khoa, T.A.; Man, M.M.; Nguyen, T.Y.; Nguyen, V.; Nam, N.H. Smart agriculture using IoT multi-sensors: A novel watering management system. *J. Sens. Actuator Netw.* **2019**, *8*, 45. [\[CrossRef\]](#)
29. Aafreen, R.; Neyaz, S.Y.; Shamim, R.; Beg, M.S. An IoT based system for telemetry and control of Greenhouse environment. In Proceedings of the 2019 International Conference on Electrical, Electronics and Computer Engineering (UPCON), Aligarh, India, 8–10 November 2019.
30. Alves, R.G.; Souza, G.; Maia, R.F.; Tran, A.L.H.; Kamienski, C.; Soininen, J.P.; Aquino, P.T.; Lima, F. A digital twin for smart farming. In Proceedings of the 2019 IEEE Global Humanitarian Technology Conference (GHTC), Seattle, WA, USA, 17–20 October 2019.
31. Verdouw, C.; Tekinerdogan, B.; Beulens, A.; Wolfert, S. Digital twins in smart farming. *Agric. Syst.* **2021**, *189*, 103046. [\[CrossRef\]](#)
32. Angin, P.; Anisi, M.H.; Göksel, F.; Gürsoy, C.; Büyükgülçü, A. AgriLoRa: A digital twin framework for smart agriculture. *J. Wirel. Mob. Netw. Ubiquitous Comput. Dependable Appl.* **2020**, *11*, 77–96.
33. Chaux, J.D.; Sanchez-Londono, D.; Barbieri, G. A digital twin architecture to optimize productivity within controlled environment agriculture. *Appl. Sci.* **2021**, *11*, 8875. [\[CrossRef\]](#)
34. González, J.P.; Sanchez-Londoño, D.; Barbieri, G. A Monitoring Digital Twin for Services of Controlled Environment Agriculture. *IFAC-PapersOnLine* **2022**, *55*, 85–90. [\[CrossRef\]](#)
35. Durão, L.F.C.; Haag, S.; Anderl, R.; Schützer, K.; Zancul, E. Digital twin requirements in the context of industry 4.0. In *Product Lifecycle Management to Support Industry 4.0, Proceedings of the 15th IFIP WG 5.1 International Conference (PLM 2018)*, Turin, Italy, 2–4 July 2018; Springer: Cham, Switzerland, 2018.
36. Ewald, R.; Uhrmacher, A.M. SESSL: A domain-specific language for simulation experiments. *ACM Trans. Model. Comput. Simul. (TOMACS)* **2014**, *24*, 1–25. [\[CrossRef\]](#)
37. Miller, J.A.; Han, J.; Hybinette, M. Using domain specific language for modeling and simulation: Scalation as a case study. In Proceedings of the 2010 Winter Simulation Conference, Baltimore, MD, USA, 5–8 December 2010.
38. Dhoubi, S.; Kchir, S.; Stinckwich, S.; Ziadi, T.; Ziane, M. A domain-specific language to design, simulate and deploy robotic applications. In Proceedings of the Simulation, Modeling, and Programming for Autonomous Robots: Third International Conference (SIMPAN 2012), Tsukuba, Japan, 5–8 November 2012.
39. Barriga, J.A.; Clemente, P.J.; Sosa-Sánchez, E.; Prieto, Á.E. SimulateIoT: Domain Specific Language to design, code generation and execute IoT simulation environments. *IEEE Access* **2021**, *9*, 92531–92552. [\[CrossRef\]](#)
40. Barriga, J.A.; Clemente, P.J.; Hernández, J.; Pérez-Toledano, M.A. SimulateIoT-FIWARE: Domain specific language to design, code generation and execute IoT simulation environments on FIWARE. *IEEE Access* **2022**, *10*, 7800–7822. [\[CrossRef\]](#)
41. Franceschini, R.; Bisgambiglia, P.A.; Bisgambiglia, P.; Hill, D. DEVS-Ruby: A Domain Specific Language for DEVS Modeling and Simulation. In Proceedings of the 2014 Symposium on Theory of Modeling & Simulation, Tampa, FL, USA, 13–16 April 2014.
42. Groeneveld, D.; Tekinerdogan, B.; Garousi, V.; Catal, C. A domain-specific language framework for farm management information systems in precision agriculture. *Precis. Agric.* **2021**, *22*, 1067–1106. [\[CrossRef\]](#)
43. Kawtrakul, A. Ontology engineering and knowledge services for agriculture domain. *J. Integr. Agric.* **2012**, *11*, 741–751. [\[CrossRef\]](#)
44. Ceh, I.; Crepinšek, M.; Kosar, T.; Mernik, M. Ontology driven development of domain-specific languages. *Comput. Sci. Inf. Syst.* **2011**, *8*, 317–343. [\[CrossRef\]](#)
45. Lamy, J.B. Owlready: Ontology-oriented programming in Python with automatic classification and high level constructs for biomedical ontologies. *Artif. Intell. Med.* **2017**, *80*, 11–28. [\[CrossRef\]](#) [\[PubMed\]](#)
46. Allocca, C.; d'Aquin, M.; Motta, E. Towards a formalization of ontology relations in the context of ontology repositories. In *Knowledge Discovery, Knowledge Engineering and Knowledge Management, Proceedings of the First International Joint Conference (IC3K 2009)*, Funchal, Portugal, 6–8 October 2009; Springer: Berlin/Heidelberg, Germany, 2009.

47. Mani, P.; Thirumalai Natesan, V. Experimental investigation of drying characteristics of lima beans with passive and active mode greenhouse solar dryers. *J. Food Process Eng.* **2021**, *44*, e13667. [\[CrossRef\]](#)
48. Baglivo, C.; Mazzeo, D.; Panico, S.; Bonuso, S.; Matera, N.; Congedo, P.M.; Oliveti, G. Complete greenhouse dynamic simulation tool to assess the crop thermal well-being and energy needs. *Appl. Therm. Eng.* **2020**, *179*, 115698. [\[CrossRef\]](#)
49. Peña-Fernández, A.; Colón-Reynoso, M.A.; Mazuela, P. Geometric analysis of greenhouse roofs for energy efficiency optimization and condensation drip reduction. *Agriculture* **2024**, *14*, 216. [\[CrossRef\]](#)
50. Wangkahart, S.; Junsiri, C.; Srichat, A.; Poojeera, S.; Laloon, K.; Hongtong, K.; Bouphe, P. Using Greenhouse Modelling to Identify the Optimal Conditions for Growing Crops in Northeastern Thailand. *Math. Model. Eng. Probl.* **2022**, *9*, 1648–1658. [\[CrossRef\]](#)
51. Łysiak, G.P.; Szot, I. The use of temperature based indices for estimation of fruit production conditions and risks in temperate climates. *Agriculture* **2023**, *13*, 960. [\[CrossRef\]](#)
52. Lata, S.; Verma, H.K. Selection of number and locations of temperature and luminosity sensors in intelligent greenhouse. *Int. J. Appl. Res.* **2018**, *13*, 10965–10971.
53. Lee, S.Y.; Lee, I.B.; Yeo, U.H.; Kim, R.W.; Kim, J.G. Optimal sensor placement for monitoring and controlling greenhouse internal environments. *Biosyst. Eng.* **2019**, *188*, 190–206. [\[CrossRef\]](#)
54. Ajani, O.S.; Aboyeji, E.; Mallipeddi, R.; Uyeh, D.D.; Ha, Y.; Park, T. A genetic programming-based optimal sensor placement for greenhouse monitoring and control. *Front. Plant Sci.* **2023**, *14*, 1152036. [\[CrossRef\]](#) [\[PubMed\]](#)
55. Wagg, D.J.; Worden, K.; Barthorpe, R.J.; Gardner, P. Digital twins: State-of-the-art and future directions for modeling and simulation in engineering dynamics applications. *ASCE-ASME J. Risk Uncertain. Eng. Syst. Part B Mech. Eng.* **2020**, *6*, 030901. [\[CrossRef\]](#)
56. Agalianos, K.; Ponis, S.T.; Aretoulaki, E.; Plakas, G.; Efthymiou, O. Discrete event simulation and digital twins: Review and challenges for logistics. *Procedia Manuf.* **2020**, *51*, 1636–1677. [\[CrossRef\]](#)
57. Jahić, B.; Guelfi, N.; Ries, B. SEMKIS-DSL: A domain-specific language to support requirements engineering of datasets and neural network recognition. *Information* **2023**, *14*, 213. [\[CrossRef\]](#)
58. Han, Z.; Qazi, S.; Werner, M.; Devarajegowda, K.; Ecker, W. On Self-Verifying DSL Generation for Embedded Systems Automation. In Proceedings of the 24th MBMV Workshop 2021, Virtual Event, Germany, 18–19 March 2021.
59. Amdah, L.; Anwar, A. A DSL for collaborative business process. In Proceedings of the 2020 International Conference on Intelligent Systems and Computer Vision (ISCV), Fez, Morocco, 9–11 June 2020.
60. Huisman, M.; Wijs, A. Model Checking Algorithms. In *Concise Guide to Software Verification: From Model Checking to Annotation Checking*; Springer International Publishing: Cham, Switzerland, 2023; pp. 79–106.
61. Pollak, D.; Layka, V.; Sacco, A. *Beginning Scala 3: A Functional and Object-Oriented Java Language*, 3rd ed.; Apress: Berkeley, CA, USA, 2022; pp. 237–245.
62. Wang, B.; Wang, Z.; Wang, X.; Cao, Y.; Saurous, R.A.; Kim, Y. Grammar prompting for domain-specific language generation with large language models. *Adv. Neural Inf. Process. Syst.* **2024**, *36*, 1–26.
63. Poltronieri, I.; Zorzo, A.F.; Bernardino, M.; de Borja Campos, M. Usa-DSL: Usability evaluation framework for domain-specific languages. In Proceedings of the 33rd Annual ACM Symposium on Applied Computing 2018, Pau, France, 9–13 April 2018.
64. Nielsen, J. Usability inspection methods. In Proceedings of the Conference Companion on Human Factors in Computing Systems, Boston, MA, USA, 24–28 April 1994.
65. Kahraman, G.; Bilgen, S. A framework for qualitative assessment of domain-specific languages. *Softw. Syst. Model.* **2015**, *14*, 1505–1531. [\[CrossRef\]](#)
66. Alaca, O.F.; Tezel, B.T.; Challenger, M.; Goulão, M.; Amaral, V.; Kardas, G. AgentDSM-Eval: A framework for the evaluation of domain-specific modeling languages for multi-agent systems. *Comput. Stand. Interfaces* **2021**, *76*, 103513. [\[CrossRef\]](#)

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.