mathematics

Review

Modeling Languages for Internet of Things (IoT) Applications: A Comparative Analysis Study

Sadik Arslan [1,2], Mert Ozkaya [3] and Geylani Kardas [1,*]

1 International Computer Institute, Ege University, Izmir 35100, Turkey
2 FIS Systems, Software & Functions Group, BorgWarner Co., Ltd., Izmir 35410, Turkey
3 Computer Engineering Department, Yeditepe University, Istanbul 34755, Turkey
* Correspondence: geylani.kardas@ege.edu.tr

Abstract: Modeling languages have gained ever-increasing importance for the Internet of Things (IoT) domain for improving the productivity and quality of IoT developments. In this study, we analyzed 32 different modeling languages that have been designed for IoT software development in terms of a set of requirements that were categorized into three groups: language definition, language features, and tool support. Some key findings are as follows: (1) performance is the most supported quality property (28%); (2) most languages offer a visual notation set only, while 6% provide both textual and visual notation sets; (3) most languages (88%) lack formally precise semantic definitions; (4) most languages (94%) support the physical, deployment, and logical modeling viewpoints, while the behavior, logical, and information viewpoints are rarely supported; (5) almost none of the languages enable extensibility; (6) Java (34%) and C (21%) are the most preferred programming languages for model transformation; (7) consistency (77%) and completeness (64%) are the most supported properties for the automated checking of models; and (8) most languages (81%) are not supported with any websites for sharing case studies, source code, tools, tutorials, etc. The analysis results can be useful for language engineers, practitioners, and tool vendors for better understanding the existing languages for IoT, their weak and strong points, and IoT industries' needs in future language and modeling toolset developments.

Keywords: Internet of Things; modeling; modeling languages; modeling toolset; practitioners

MSC: 68M11; 68N01; 68N15; 97P30; 97P40; 97P50

check for updates

Citation: Arslan, S.; Ozkaya, M.; Kardas, G. Modeling Languages for Internet of Things (IoT) Applications: A Comparative Analysis Study. *Mathematics* **2023**, *11*, 1263. https://doi.org/10.3390/math11051263

Academic Editor: János Sztrik

Received: 30 January 2023
Revised: 3 March 2023
Accepted: 4 March 2023
Published: 6 March 2023

1. Introduction

The Internet of Things (IoT) is a network of physical objects and virtual devices that communicate and interact with each other. "Things" here represent such things as wearable devices, physical sensors (e.g., temperature and humidity sensors), engines, RFID tags, and modern self-driving cars fully equipped with computers. Things are interconnected and can exchange data. Devices in the IoT can be monitored and controlled remotely [1].

IoT opens up great opportunities to improve the way we live and work across heterogeneous things that can be either computer-based systems or physical objects. IoT promotes the seamless and dynamic collaboration among things. However, designing, developing, deploying, and managing a living network of things at runtime requires facing many challenges to do with the heterogeneity, complexity, and availability of devices and large amounts of data, security, and privacy [2–4].

As revealed in empirical studies and the surveys on IoT, developing IoT applications is considered a complex, time-consuming, and challenging activity. Udoh and Kotonya [5] showed in their study that practitioners face challenges while designing, developing, and maintaining IoT software systems. Noura et al. [6] pointed out the lack of support for domain-specific modeling, interoperability, and automation such as code generation and test generation for IoT. Berrouyne et al. [7] further indicated two essential IoT domain problems: (1) incomplete software design and (2) faulty manual code generation. Grace et al. [8] pointed out the interoperability problems for IoT software systems. Ciccozzi et al. [4] emphasized the communication difficulties between stakeholders. Siegel et al. [9] indicated that the rapid growth of IoT is limited due to the lack of support for managing resource usage, privacy, and security issues. Larrucea et al. [10] analyzed the software engineering challenges and solutions for IoT domain practitioners, while Durmaz et al. [11] considered the challenges of the intermittent execution of batteryless IoT devices and proposed a new IoT programming model. Amin et al. [12] indicated many challenges in the applications of IoT in practice, which are concerned with integrating heterogeneous data between devices, data security, communication protocols, and big data collection and analysis. An overview of IoT applications within the context of big data, data science, and network science disciplines and the technologies for connecting them was also discussed in their study. Moreover, another recent trend of IoT called the Social Internet of Things (SIoT) for managing IoT application challenges such as scalability, service discovery, and heterogeneity was investigated in [13].

The problems discussed above can be targeted with Model-Driven Engineering (MDE) [14–16], which promotes the high-level specifications of abstract models for managing the complexity of systems and the use of modeling tools for the automated (i) detection of errors, (ii) decision making, and (iii) generation of useful artifacts [17–19]. The IoT problems that are summarized above can be addressed with MDE thanks to its key advantages, which are "abstraction", "separation of concerns", "high-level domain-specific solutions", "tool support for automation", and "early analysis of design decisions". Abstraction basically promotes suppressing unnecessary details that make things complex while specifying models. The separation of concerns also promotes abstraction with its support for the separation of models into concerns (e.g., structural, behavioral, interaction, concurrency, information, and development) [15]. With modeling tools (e.g., [20]), abstract models can be specified textually/visually and processed for, e.g., (i) tracing models; (ii) analyzing models for quality properties (e.g., security); (iii) making automated decisions and visualizing some findings; (iv) generating documents, code, and any useful artifacts; and (v) exporting the modeled data into some standard exchange formats.

Modeling is facilitated by the modeling languages [21–23], which provide syntax and semantics for the high-level (and sometimes precise) specifications of models and are supported with some tools for editing and processing models including model editors, model validators, and model transformers. Modeling languages can be either general-purpose or domain-specific. Unified Modeling Language (UML) [24], Systems Modeling Language (SysML) [25], Business Process Modeling Notation (BPMN) [26], and ArchiMate [27] are widely used general-purpose languages in industry for specifying any type of system. DSMLs are each used for a particular domain. For instance, AADL [28], Modelica [29], and Koala [30] are provided for the development of real-time embedded systems, cyber-physical systems [29], and consumer electronics, respectively. Metamodeling technologies (e.g., Eclipse Modeling Framework (EMF) [31] and Metaedit+ [32]) are often used in industry nowadays, facilitating the design of modeling languages and the development of their supporting toolset.

Our focus in this study is the empirical studies/research and surveys on the IoT domain, as the usage of promised solutions of MDE for the IoT has been ever-increasingly expanding [10,33,34]. Indeed, since the end of the 2010s, the use of MDE has increased, e.g., in home automation [35], production systems [36,37], health systems [38], and smart cities [39]. However, it is not so easy for the practitioners who are interested in using/extending the existing IoT modeling languages and tools to:

- Determine the existing IoT-based modeling languages and tools in the literature;
- Compare the languages for certain requirements;
- Choose the optimal approach(es) that best fits their requirements.

Moreover, by just using the existing literature, language engineers for the IoT domain cannot easily determine the strong and weak points of the existing languages with regard to the important properties for practitioners. Similarly, tool vendors cannot also analyze the existing language tools for IoT and determine their strong and weak points. Without those data, the language engineers and tool vendors cannot easily address the needs of the IoT modeling industry. While many empirical studies have already been conducted on the analysis of modeling languages, none of them specifically considers the IoT modeling languages and tools. As discussed in the related work section, the existing studies on the IoT modeling languages and tools do not intend to reveal the existing languages and analyze them for certain requirements, instead focusing on IoT development challenges with MDE, such as complexity [39,40] and heterogeneity [37,41–44].

To bridge the gaps discussed above, our goal in this study was to analyze the existing IoT-based modeling languages with a set of requirements that are considered important for practitioners. To determine the requirements according to which the languages can be analyzed, we considered Lago et al.'s seminal work [45] on language requirements, where the language requirements are categorized into three groups which are language definition, language features, and tool support. Each category of requirements is supported with a cohesive set of requirements, as introduced in [45]. Our contributions in this study include:

- Providing a single resource for accessing all the IoT modeling languages;
- The categorization of the languages based on their support for the language definition, language features, and tool support requirements;
- Determining the language and tool requirements that are popular and those that are weakly supported among the modeling languages;
- The identification of the languages that can be considered powerful in different categories of requirements.

We strongly believe that the results of this study will be highly beneficial for different stakeholders involved in IoT software development, as the current literature does not provide support for the analysis of the modeling languages for IoT and tools for the needs of the practitioners. Practitioners can find out the language(s) that best meet their needs and see if they can extend those languages or not. Language engineers and tool vendors can use the results to determine the strong and weak points of the existing modeling languages for IoT and initiate projects for bridging the gaps. Moreover, for researchers, the results can trigger new empirical studies that focus on better understanding practitioners' interest in modeling, modeling languages, and even metamodeling in the IoT field and further inspire the researchers to conduct similar analytical studies on the modeling languages developed for different domains.

This paper is organized as follows: Related work is discussed in Section 2. In Section 3, the requirements for the analysis of modeling languages for IoT are introduced. In Section 4, the research methodology in selecting modeling languages for IoT to be analyzed, collecting requirements data, and analyzing these data is explained. In Section 5, the analysis results for the language definition requirements, language feature requirements, and tool support requirements are discussed. In Section 6, we discuss the achieved results and the threats to their validity. Section 7 concludes the paper.

2. Related Work

Many studies focus on the IoT domain as one of the most promising areas of the technology today. These studies generally cover IoT development and naturally consider the complexity of IoT application development. MDE comes with suggestions to reduce this complexity. Many studies in the literature describe the language, framework, and toolkit that include MDE of IoT applications. However, almost none of these existing studies examine the needs of practitioners in terms of MDE languages or tools.

The related work is presented in two sections: (1) studies analyzing the languages and (2) MDE for IoT studies.

2.1. Surveys on Architectural and Modeling Languages

One of the most comprehensive studies in the literature that analyzes languages considering the needs of practitioners was carried out by Ozkaya [46]. The parameters that are very important for practitioners were examined in 124 different architectural languages (ALs). Information and comparison opportunities for practitioners to choose the language most suitable for their needs among ALs were provided by this study.

Survey studies with a narrower scope than [46] and focusing on the analysis of practitioners' needs are also available in the literature [47–60]. However, the number of languages studied and the language features analyzed in these studies are very limited.

Medvidovic and Taylor [47] and Clements [48] analyzed a group of ALs in order to describe the languages and provide a framework, which was made in the 1990s. Medvidovic et al. [49] later extended their previous work, which mainly focused on the technological aspect of software architectures, to include ALs in the 2000s. However, only eight languages were considered in this study. Another study was conducted by Vestal [50] which analyzed the four main languages for many criteria such as support for components and connectors, formal analysis, and automatic code generation. In the study of Hilliard and Rice [51], ALs developed in the 1990s were analyzed to measure expressiveness. In the study of Ali et al. [52], the authors focused on understanding whether languages offer explicit constructs in their notation to indicate distributed and mobile software systems and analyzed eight languages. In the study of McKenzie et al. [53], the authors focused on using formal ALs for the analysis and design of simulation systems and their key features. In Qin and Malik's [54] study, the authors surveyed eight different ALs for specification and formal analysis of retargetable compiler systems (e.g., assemblers and simulators). Mishra and Dutt [55] divided ALs into three groups: structural, behavioral, and mixed ALs. Structural ALs support logical view and behavioral ALs support behavioral view, while hybrid ones support both views simultaneously. Ozkaya and Kloukinas [56] analyzed 12 ALs for formal analysis, usability, and feasibility. Balaban et al. [57] analyzed three popular textual modeling languages to determine their similarities and differences, while Bergmayr et al. [58] examined 19 cloud modeling languages for a set of requirements that were categorized as language characteristics, modeling capabilities, and tool support. Twelve different ALs that support microservice architecture specifications with regard to six different language categories were considered in Lelovic et al. [59]. Lastly, Alidra et al. [60] analyzed 16 modeling languages that support fog computing for a comprehensive set of requirements which were categorized as language scope, language definition, implementation, capabilities, interoperability, exploitation, validation, and documentation.

Table 1 gives the analysis results of the similar studies with regard to the properties of interest, which are (i) the year the study was published, (ii) the scope of the study, (iii) the number of languages considered in the study, and (iv) 12 different requirements that are categorized as the language definition, language features, and tool support. This requirement categorization was inspired by Lago et al.'s framework [45], and all these defined requirements will be discussed extensively in the following Section 3. None of the similar studies considers all 12 requirements which our study uses for analyzing the modeling languages. The only exception here is Ozkaya [46], which, however, focuses on the ALs in general, not the IoT domain. Our study differs from the previous ones by both taking into consideration all of Lago et al.'s [44] language criteria enriched with the new specifications for IoT application development requirements and multiple viewpoints and utilizing these criteria for evaluating modeling languages for the IoT domain for the first time.

Table 1. The summary of the analysis of the related studies.

Survey	RY	Scope	Cons Lang#	Language Definition			Language Features			Tool Support					
				NonFunc Req	NotSet	ForSem	MulView	ExMec	Prog Frame	Auto Analy	LVM	Col	Ver	Know Man	S-C
Our study	2023	IoT-based modeling languages	32	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Alidra et al. [60]	2023	Fog computing	16	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes	Yes
Lelovic et al. [59]	2022	Microservice architectures	12	No	Yes	No	No	No	Yes	No	No	No	No	No	No
Bergmayr et al. [58]	2018	Cloud modeling languages	19	No	Yes	Yes	Yes	No	Yes	Yes	Yes	No	No	No	Yes
Ozkaya [46]	2018	Defining an AL's capabilities	124	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Balaban et al. [57]	2016	Textual modeling languages	3	No	Yes	Yes	Yes	No	No	Yes	Yes	No	No	No	No
Ozkaya and Kloukinas [56]	2013	Defining an AL's capabilities	13	No	No	Yes	No	No	No	Yes	Yes	No	No	No	No
Ali et al. [52]	2008	Distributed and mobile software systems	8	No	Yes	Yes	No	No	Yes	Yes	Yes	No	No	No	Yes
Qin and Malik [54]	2007	Retargetable compilers	12	No	No	Yes	No	Yes	Yes	Yes	Yes	No	No	No	Yes
Medvidovic et al. [49]	2007	Defining an AL's capabilities	9	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	Yes
Mishra and Dutt [55]	2005	Programmable embedded systems	9	No	No	No	No	No	Yes	Yes	No	No	No	No	Yes
McKenzie et al. [53]	2004	Simulation systems	2	No	Yes	Yes	Yes	No	Yes	Yes	Yes	No	No	No	Yes
Medvidovic and Taylor [47]	2000	Defining an AL's capabilities	9	Yes	No	Yes	Yes	No	Yes	Yes	Yes	No	Yes	No	No
Hilliard and Rice [51]	1998	Defining an AL's capabilities	13	No	No	No	Yes	No	No	No	No	No	No	No	Yes
Clements [48]	1996	Defining an AL's capabilities	8	No	Yes	Yes	No	Yes	Yes	Yes	No	No	No	No	Yes
Vestal [50]	1993	Defining an AL's capabilities	4	Yes	Yes	Yes	No	Yes	Yes	Yes	Yes	No	Yes	No	Yes

Abbreviations: Auto Analy: automated analysis, Col: collaboration, Cons Lang#: the number of considered languages, ExMec: extension mechanism, ForSem: formal semantics, Know Man: knowledge management, LVM: large-view management, MulView: multiple viewpoints, NonFunc Req: nonfunctional requirements, NotSet: notation set, Prog Frame: programming framework, RY: release year, S-C: software architecture-centric design, Ver: versioning.

Additionally, it is worth noting that recently significant research also exists on surveying modeling languages and MDE for various domains. For instance, de Araujo Silva et al. [61] discussed the motivations, challenges, methods, and future perspectives of MDE in robotics, while [62] presented 28 tools which assist modeling, model transformation, and model checking during MDE of embedded systems. Again for the embedded system domain, Liebel et al. [63] provided information about the methods and tools used, purposes of the models, effects of using it, and weaknesses of model-based software engineering. Visual programming tools for the development of intelligent and social robots were investigated in [64], and the need for the validation of these tools in real environments instead of laboratory settings was highlighted. Finally, modeling aspects of cyber-physical systems from the perspectives of, e.g., security engineering [65], MDE techniques used [66], multi-paradigm modeling [67], and applied tools and the research practices [68] were also considered. Our study contributes to these existing modeling surveys by introducing a comparative evaluation with architectural language requirements and providing an analysis of a different domain that covers modeling IoT applications.

2.2. MDE for IoT Studies

The use of MDE techniques to design and implement IoT systems has become one of the most popular topics in both IoT and software engineering research fields. For example, Li et al. [69] identified the IoT trends in the field of space sciences and discussed how the sensors and functional IoT infrastructures of digital twins can be modeled. Fortino et al. [70] aimed at providing the baseline definitions for IoT development products such as methodologies, frameworks, platforms, and tools and indicated the benefits of using model-driven IoT development methodologies especially to tackle the difficulties of supporting interoperability for the target IoT systems. Silva and Hirmer [71] surveyed several IoT environment models according to their maturity, hierarchy, availability and implementation, and geolocation support to describe the components of IoT systems, such as devices, attached sensors, and actuators. Although IoT environment models can naturally be utilized, e.g., as the metamodels of current or future IoT modeling languages, Silva and Hirmer's study did not cover these modeling languages and their evaluation. Starting from the heterogeneous aspects of IoT systems' data, communication, and implementation layers, Ihirwe et al. [72] aimed to determine the implications of the current trend of moving traditional system modeling infrastructures to the cloud. They focused on different cloud-based IoT system development approaches and highlighted the benefits of adopting cloud-based modeling of IoT systems in attracting more citizen developers, collaborative modeling, productivity, maintenance, and monitoring and debugging IoT applications.

To simulate the communication between IoT devices and their behavior in different situations, model-based development of a smart home scenario was discussed in Kölsch et al. [35]. The modeling approach presented in [36] enabled the developers to design IoT components using the UML and SysML standards. A UML profile was also proposed to automate the transformation of the IoT component to an interface ready to be integrated into the modern IoT manufacturing environment. Likewise, Khaeel et al. [37] introduced model-driven development tools as part of their IoT framework to develop heterogeneous applications for the car manufacturing industry. The use of such modeling tools enabled them to build IoT applications, linking their components and implementing proper security solutions. Mezghani et al. [38] demonstrated the efficiency of using modeling patterns by developing a cognitive monitoring system for managing patients' health based on heterogeneous wearables. Hause et al. [39] discussed how MDE tools can assist implementing and executing IoT system designs for smart cities by especially focusing on the selection of the controlled or coordinated traffic signals for the predictive and adaptive traffic management systems. A model-driven and service-centric approach was presented in [41] to cope with the challenges in both the convergence of different network communication protocols and IoT and the uniform management of IoT services. Sharaf et al. [42] introduced a code generation methodology that covers a full chain of modeling, analyzing, and implementing IoT systems. Similarly, in Pusztai et al.'s methodology [43], design-time modeling of IoT device interfaces was possible, and code for participating devices was automatically generated from these design models.

Although the above-mentioned studies provided valuable MDE methods and/or tools to be employed in current IoT applications, they did not provide a comprehensive analysis of the language requirements and features of the modeling languages that exist for IoT development as we do in our study.

3. The Requirements for the Modeling Languages

The approach proposed by Lago et al. [45] was used in this study, which offers a framework for designing and developing languages with the considerations of practitioners' needs and expectations. Indeed, the framework here was developed after conducting surveys on 48 different practitioners from 40 IT companies and thus is the result of feedback from practitioners' expectations [73]. In this framework, language requirements are divided into three main groups: (1) language definition, (2) language features, and (3) tool support. Information about these groups is given in the following subsections.

3.1. Language Definition

Language definition can be handled under three main groups: (1) nonfunctional properties, (2) textual and visual notation sets, and (3) formal semantics. These relate to the definition of language syntax and semantics. Nonfunctional properties are to do with specifying quality requirements (e.g., security, performance, and reliability) and verifying models against those requirements. Any language can offer textual, visual (also known as graphical), or hybrid notation sets. Semantics can be defined either formally or informally. Formal semantics for a language is defined using mathematical-based formal techniques such as π-computation [74] and Communicating Sequential Processes (CSP) [75]. Informal semantics are often described using plain English.

3.2. Language Features

Language features are considered in terms of three requirements: (1) multiple viewpoints, (2) extensibility and customization, and (3) programming framework. Multiple viewpoint support is to do with the separation of concerns for managing the complexity of software systems. Each viewpoint focuses on a particular concern and enables modeling for that concern only (e.g., separating models in accordance with the logical, information, physical, deployment, behavior, concurrency, development, and operational viewpoints) [76–78]. Extensibility and customization are to do with extending languages with some desired capabilities such as domain-specific elements, analytics support, and support for different views. In this study, two different extension mechanisms, syntax and semantic extensions, are discussed [79]. The programming framework is concerned with the support for any framework that can facilitate the specification, analysis, and transformation of software models.

3.3. Tool Support

Six different requirements are addressed for the tool support. These are: (1) automated analysis, (2) large-view management, (3) collaboration support, (4) versioning support, (5) knowledge management support, and (6) software architecture-centric design support. The automated analysis support is considered in terms of analyzing models for four important goals: completeness, consistency, correctness, and compatibility. Additionally, other important analysis capabilities are also considered such as exhaustive model checking, simulation, and user-defined requirements. Large-view management is concerned with the techniques and methods used for modeling large and complex views of software systems in a more manageable way that is easier to understand and analyze, such as sub-diagramming support and composite elements. The collaboration support is concerned with the multi-user support for the specifications of models. Indeed, the physically separated users may need to specify models synchronously or asynchronously. Versioning support is about keeping a repository of software specifications with some release details. Knowledge management support relates to publishing language-related information such as tools and case studies, publications, and user guides through websites. It also provides platforms such as discussion forums for practitioners to exchange ideas with each other and find solutions to their language-related problems. The architecture-centric design relates to the tools for integrating the software architecture design with other phases of software development, including requirements specification and analysis, low-level software design, and software implementation.

4. Research Methodology

In this study, we used the popular search engines Google and Google Scholar to find any documents regarding the modeling languages for IoT. All publications from our accounts such as IEEE/ACM/Springer-link/Elsevier/Wiley were reviewed and recorded between 15 April and 11 August 2022. All languages and tools, if any, and websites were examined; all available technical reports, theses, toolkits, and user manuals were kept in an organized manner. A folder consisting of a series of subfolders was created for each language to be analyzed. First of all, the web address of the language (if any) was saved in one of the subfolders. Note that each language was searched with Google to access any website that reveals the language's supporting materials. Language-related publications that were available in IEEE/ACM/Springer-link/Elsevier/Wiley were stored in another subfolder. Technical reports/dissertations that can be downloaded for free were also included in the subfolder. In another subfolder, the supporting toolset (if any) was downloaded to experiment with the language. The language manuals (if any) were taken and stored in a subfolder.

To collect the published papers about the modeling languages for IoT, we followed the screening and reviewing process depicted in Figure 1 which is based on the Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) guideline [80]. In the first stage (identification), we scanned the digital databases (IEEE, ACM, Springer-link, Elsevier, and Wiley) that enable access to the journal and conference publications and found 495 papers about the modeling languages for IoT. To obtain the papers (i.e., records in Figure 1), we used many keywords on Google, and some important ones are as follows: (1) modeling languages IoT; (2) IoT-based modeling languages; (3) IoT modeling; (4) model-based (and -driven) engineering in IoT; (5) metamodeling IoT; (6) survey on IoT modeling; (7) practitioners modeling IoT; and (8) analysis of modeling languages IoT. Having obtained 495 different records with the keywords given above, we then examined and filtered the records whose duplicates are available in our list. So, we removed 172 duplicate records whose titles were the same as some other existing records but had been published in different databases. We ended up with 323 records that could then be considered as reports, each of which represented a document supplying information about a particular study. Reports could be a journal article, preprint, conference abstract/paper, study register entry, dissertation, unpublished manuscript, or any other document providing relevant information. On the other hand, a record is the title or abstract (or both) of a report indexed in a database or a website.

In the next stage, among the 323 reports remaining, we removed another 81 reports that lacked full texts and ended up with 242 reports (i.e., papers with full texts). Then, we applied a set of exclusion criteria that we believe are highly important for obtaining the sub-list of reports which we could effectively use for our study. These exclusion criteria and the number of the studies excluded according to each of these criteria were as follows: (1) Study is a secondary study (survey, systematic mapping, systematic review, etc.) (n = 29). (2) Study is irrelevant to IoT or any of its application domains and the field of software modeling (n = 86). (3) Study is a summarized version of a complete work already in the search pool (n = 24). (4) Study is a kind of educational, editorial, tutorial, or other material (i.e., not a scientific paper) (n = 30). (5) Study is not written in English (n = 34). After excluding the reports based on the exclusion criteria, we ended up with 46 reports and 32 studies. These 46 reports related to 32 different modeling languages. So, in our study, we decided to focus on these 32 languages.

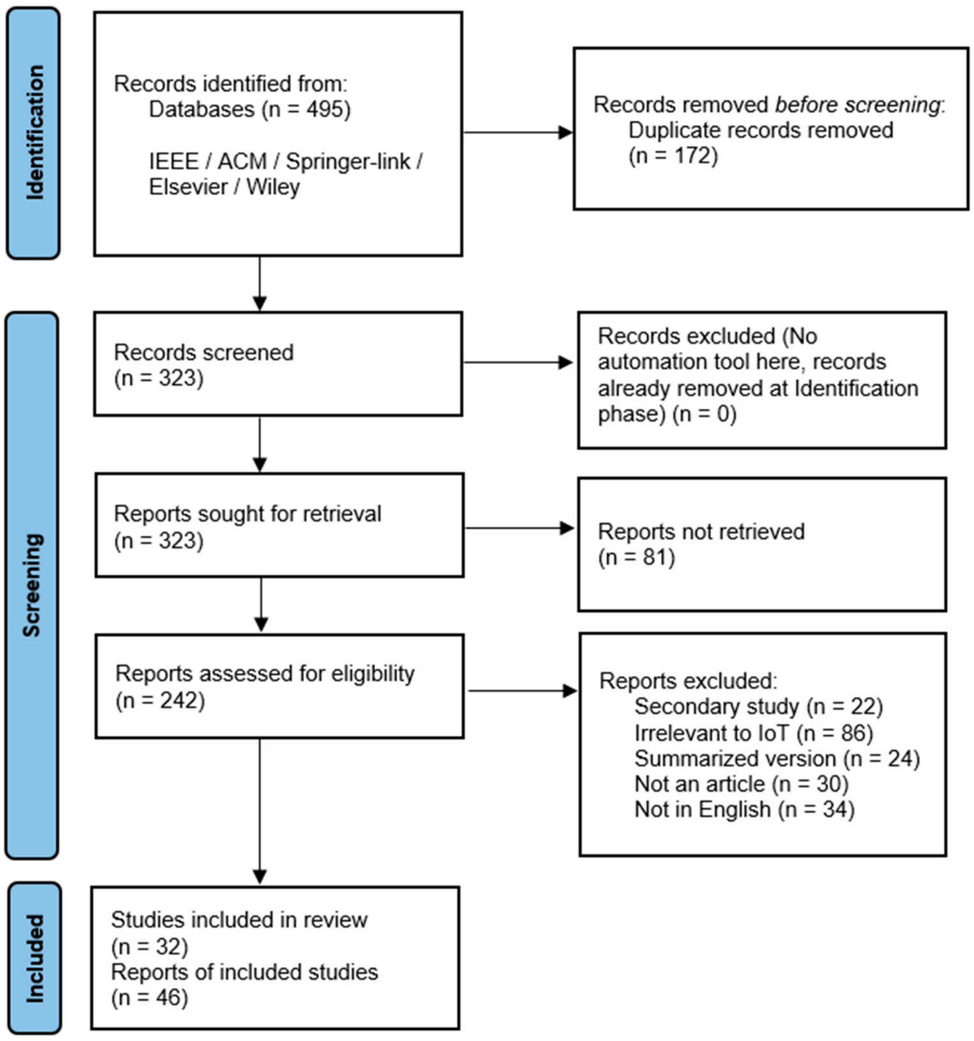
Identification
Records identified from:
Databases (n = 495)
IEEE / ACM / Springer-link / Elsevier / Wiley
Records removed *before screening*:
Duplicate records removed
(n = 172)
Screening
Records screened
(n = 323)
Records excluded (No automation tool here, records already removed at Identification phase) (n = 0)
Reports sought for retrieval
(n = 323)
Reports not retrieved
(n = 81)
Reports assessed for eligibility
(n = 242)
Reports excluded:
Secondary study (n = 22)
Irrelevant to IoT (n = 86)
Summarized version (n = 24)
Not an article (n = 30)
Not in English (n = 34)
Included
Studies included in review
(n = 32)
Reports of included studies
(n = 46)

Figure 1. The flow-chart diagram for our screening and reviewing process of the collected papers on the IoT modeling languages using the PRISMA guideline [80].

After reviewing and filtering the collected papers, we started the data collection process for each of the 32 modeling languages considered and thus examined the related papers based on a set of requirements. To determine the set of requirements here, we focused on the language requirements framework proposed in Lago et al.'s seminal study [45]. Lago et al. categorized the language requirements as language definition, language features, and tool support and proposed a set of requirements for each category. To enhance the precision of our data analysis, we extended the requirements proposed by Lago et al. with some sub-requirements as shown in Figure 2. For instance, the multiple viewpoints support was considered in our study in terms of the support for some specific modeling viewpoints that we believe are highly crucial for IoT software development, which include logical, information, physical, deployment, behavior, concurrency, development, and operational viewpoints [76]. The analysis tool support was also considered in terms of the support for some precise analysis goals (i.e., completeness, consistency, correctness, and compatibility) [77] and some specific analysis capabilities (e.g., deadlock checking and simulation support). We analyzed each of the 32 languages for each requirement and their sub-requirements (if any) listed in Figure 2 and collected data accordingly.

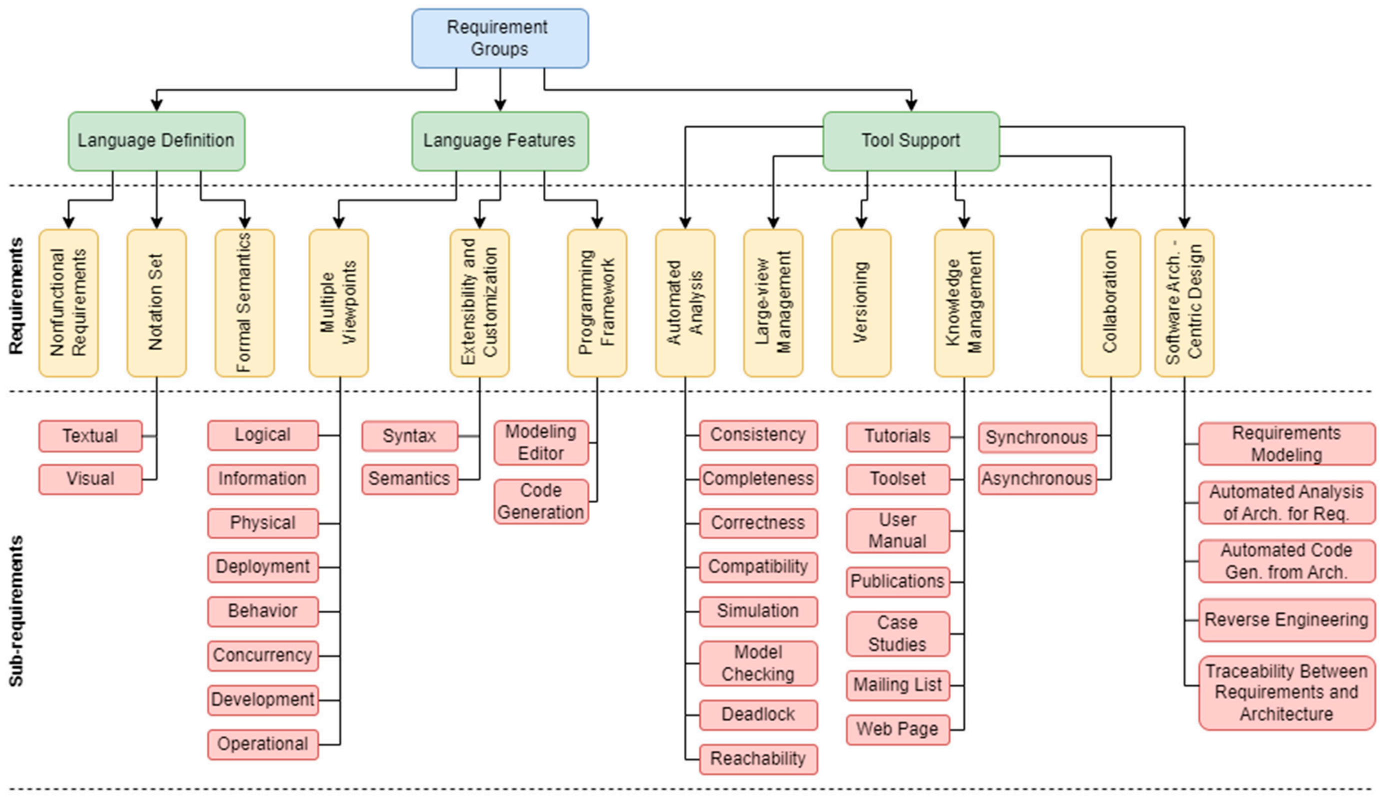

Figure 2. The requirements classification diagram that we used for analyzing the modeling languages for IoT.

To maximize the quality of the data collected, each author of this study performed the data collection process individually. So, each author examined the papers (and any other helper documents and tools) for the 32 modeling languages and obtained the data with regard to the requirements given in Figure 2. Then, the collected data by the authors were compared to determine any inconsistencies. The inconsistent parts were re-analyzed together with all authors until reaching a consensus.

To collect data for a modeling language based on a set of requirements, several iterations were performed by the authors. In the first iteration, the papers of each language were reviewed by the authors to familiarize themselves with the language (i.e., its definition, features, and tool support). In cases where theoretical knowledge was not very helpful in understanding the language and a toolkit was available, the toolkit was loaded. The language was tested with simple case studies such as reading from a temperature sensor. Next, a table was created in an Excel sheet by the authors to store the data to be collected for the languages, with columns for the requirements and rows for the corresponding languages. In the next iteration, the publications of each language (and also any helper documents and tools if needed) were reviewed for the set of requirements considered, and the data collected for each language were stored in the Excel table. The Excel table was also checked in another iteration to determine if there were missing data for each language.

After collecting the data and analyzing them together with the authors for any inconsistencies, we focused on analyzing the data to come up with some results and lessons. To this end, we analyzed the data collected for each requirement separately and we also focused on any correlations where satisfying one requirement could affect the level of consideration for another related requirement. While analyzing the data, we re-examined the papers for some languages when we were unsure about the completeness and correctness of the data we had. Next, we drew some charts using MS Office Excel and SPSS tools to visualize the results for the requirements. We prepared tables for the requirements where we displayed information regarding the languages' support for the requirements.

The tables give information about which languages support which requirement(s) and how. We also drew pie charts to display the percentages of the languages that support the requirements with particular attributes (i.e., the division of the number of languages supporting one particular attribute by the number of languages supporting the requirement in general) so as to show which attributes of the requirement are more/less popular among the languages (e.g., the programming languages' support of the languages).

5. Analysis Results

In this section, we discuss the analysis results according to the language requirements, introduced in the previous section, and categorized as the language definition, language features, and tool support.

5.1. Language Definition

5.1.1. Nonfunctional Properties

Nonfunctional properties define the quality requirements for a software system such as performance and security requirements [15]. Nonfunctional requirement properties can also be specified during the architectural design of software systems and further used for analyzing models.

As shown in Table 2, many modeling languages for IoT support the specifications of one or more nonfunctional properties such as performance, safety, reliability, availability, plannability, and resource consumption. Figure 3 shows the percentages of the languages that support each type of nonfunctional requirements indicated in Table 2 (i.e., the division of the number of languages that support each type of nonfunctional property by the number of languages supporting the nonfunctional properties). The most supported nonfunctional properties are performance (28%), security (14%), and reliability (10%). Note that 24% of the languages support the specifications of any type of nonfunctional properties via the use of aspects and formal methods.

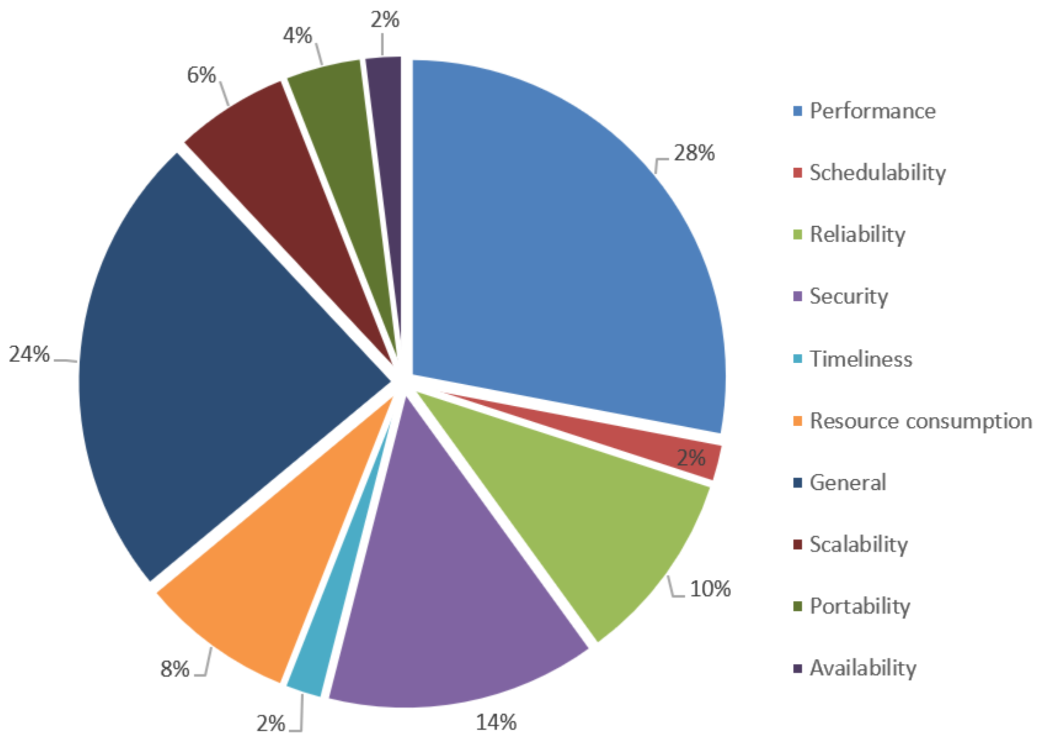

Figure 3. The percentages of the modeling languages for IoT that support nonfunctional properties.

Table 2. The modeling languages for IoT that support the different types of nonfunctional properties and their notation set types.

Language	Nonfunctional Requirements	Notation
DSML4TinyOS [81–83]	Performance	Graphical notation
ThingML [22,84–86]	Performance, schedulability, reliability	Graphical notation
UML4IoT [36,87,88]	Performance, reliability, security, timeliness, resource consumption	Graphical notation
IoT-PML [89]	General	Graphical notation
Centurión et al. [90]	Security, scalability	Graphical notation
COMFIT [91]	Performance, reliability, security, portability, resource consumption	Graphical notation
HSML [92]	Performance, resource consumption	Hybrid (FSM- and HTML-like)
IoTDraw and SoaML4IoT [93,94]	General	Graphical notation
Alulema et al. [44]	Performance	Graphical notation
EL4IoT [95,96]	General	Graphical notation
Asici et al. [97]	Performance	Graphical notation
Sosa-Reyna et al. [98,99]	General	Graphical notation
Genesis [100,101]	General	Graphical notation
BRAIN-IoT [102]	Performance, security, resource consumption	Graphical notation
IADev [103]	Performance, reliability, scalability	Graphical notation
Iovino et al. [104]	Security, availability	Graphical notation
IoTSuite [105,106]	Scalability	Graphical notation
Vitruvius [107]	General	Graphical notation
Midgar [108,109]	Performance	Graphical notation
Schachinger and Kastner [110]	General	Graphical notation
Karkouch et al. [111]	General	Graphical notation
IoTLink [37,112,113]	Performance	Graphical notation
SysML4IoT [114,115]	General	Graphical notation
FRASAD [116]	Portability	Graphical notation
Vorto [117]	General	Textual notation
DivEnact [118]	Reliability	Graphical notation
ENACT DevOps [119]	General	Graphical notation
FogUML2Code [43]	Security	Graphical notation
Betancourt et al. [120]	General	Graphical notation
MAF [121]	Performance	Graphical notation
Hassine et al. [122]	Performance	Graphical notation
SecKit [123]	Performance, security	Tree notation

5.1.2. Textual or Visual Notation Set

The textual modeling notation set for a language promotes writing formatted text in accordance with the language syntax definitions, while the visual notation set for a language promotes drawing diagrams (sometimes supplemented with text) using a set of language concepts represented with visual symbols. Modeling languages for IoT offer textual, visual, or hybrid notation sets, which could affect the practitioners' choice.

Table 2 shows the languages with their supported notation set types. Note that HSML is the only language that offers a hybrid notation set. HSML essentially supports multiple-viewpoint modeling where different viewpoints are supported with different types of notation sets. Vorto is the only language that supports a textual notation set exclusively. SecKit is supported with a tree notation. As also indicated in Figure 4, all other modeling languages for IoT support a visual notation set exclusively.

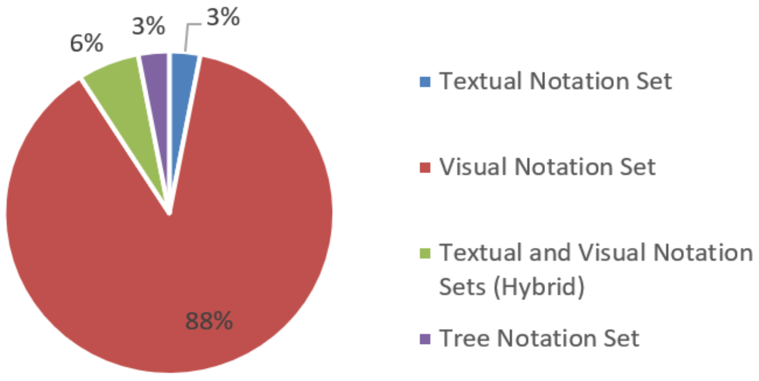

Figure 4. The percentages of the modeling languages for IoT that offer different types of notation sets.

5.1.3. Semantics Definition

The semantics of a language can be defined informally or formally. The informal semantics of a language are defined in simple English. Therefore, the language notation set with informally defined semantics may not always be clearly understood. Indeed, despite being the most used language in industry, UML [24] has no formally defined semantics, and thus UML diagrams may sometimes cause ambiguities and different interpretations. The formal semantics of a language are fully defined using mathematically based formal methods, which lead to precise model specifications that are ensured to be interpreted in the same way. Such formal methods can take different forms, namely, operational [124], denotational [125], and axiomatic [126]. Moreover, the semantics of some languages are defined using formal specification languages such as π-calculus [74], Petri nets [127], and Temporal Logic [128].

The analysis results show that most modeling languages for IoT are supported with informal semantic definitions (88%). This result is similar to (or even slightly greater than) the findings of [129] where 83% of domain-specific languages were informally defined. It is worth noting that languages which provide some sort of translational semantics over a series of model-to-model and/or model-to-text transformations to execute the IoT models on underlying implementation platforms are also considered as having informal semantics since they still lack a formal definition of these semantics. As can be seen from Table 3, 29 languages have informally defined semantics, 1 uses fUML [130], 1 uses Petri nets, and 1 uses Temporal Logic. It is expected that the informal semantics ratio is high for the highly heterogeneous IoT domain, which contains many different units.

5.2. Language Features

5.2.1. Multiple Viewpoints

In this study, logical, information, physical, deployment, behavior, concurrency, development, and operational multiple viewpoints proposed by Taylor et al. [77] are considered. The logical viewpoint describes the logical components and connectors that together compose a software system. The information viewpoint describes how components store, manipulate, and exchange their data. The physical viewpoint describes the hardware components on which the software components will be placed and the physical relationships between the hardware components. The deployment viewpoint defines the mapping between hardware components and logical components. The behavior viewpoint describes the behavior of system components and any complex interaction mechanisms (i.e., interaction protocols). The concurrency viewpoint describes the strategies and decisions in managing the concurrency issues such as threading and synchronization between components, deadlock, and race condition. The development viewpoint describes how to implement and test components and connectors. Finally, the operational viewpoint describes how to run the system in the environment of the specified software system, including plans to set up the system, any configuration, support for users, system monitoring, and backup/restore issues.

Table 3. The modeling languages for IoT that support informal semantics or formal semantics definition.

Language	Formal Semantics
DSML4TinyOS [81–83]	Informal semantics
ThingML [22,84–86]	Informal semantics
UML4IoT [36,87,88]	Informal semantics
IoT-PML [89]	Informal semantics
Centurión et al. [90]	Informal semantics
COMFIT [91]	Informal semantics
HSML [92]	Informal semantics
IoTDraw and SoaML4IoT [93,94]	fUML [130]
Alulema et al. [44]	Informal semantics
EL4IoT [95,96]	Informal semantics
Asici et al. [97]	Petri nets [127]
Sosa-Reyna et al. [98,99]	Informal semantics
Genesis [100,101]	Informal semantics
BRAIN-IoT [102]	Informal semantics
IADev [103]	Informal semantics
Iovino et al. [104]	Informal semantics
IoTSuite [105,106]	Informal semantics
Vitruvius [107]	Informal semantics
Midgar [108,109]	Informal semantics
Schachinger and Kastner [110]	Informal semantics
Karkouch et al. [111]	Informal semantics
IoTLink [37,112,113]	Informal semantics
SysML4IoT [114,115]	Informal semantics
FRASAD [116]	Informal semantics
Vorto [117]	Informal semantics
DivEnact [118]	Informal semantics
ENACT DevOps [119]	Informal semantics
FogUML2Code [43]	Informal semantics
Betancourt et al. [120]	Informal semantics
MAF [121]	Informal semantics
Hassine et al. [122]	Informal semantics
SecKit [123]	Temporal Logic [123]

According to the results, almost all the languages support physical (94%), deployment (94%), and logical (94%) viewpoints. Given that the IoT domain is tightly dependent on hardware and software distribution, it is quite normal to have a very high rate of physical and deployment viewpoint support. The rest of the viewpoints considered are rarely supported by the languages: 34% of the languages support the behavior viewpoint and 3% of the languages support the information viewpoint. Table 4 shows the languages with multiple viewpoints support. The language that provides the widest viewpoint support is ENACT DevOps [119], which supports physical, behavior, logical, information, and deployment viewpoints.

Table 4. The modeling languages for IoT that support multiple viewpoints.

Language	Multiple Viewpoints
DSML4TinyOS [81–83]	Physical, deployment, logical
ThingML [22,84–86]	Physical, logical, behavior, deployment
UML4IoT [36,87,88]	Physical, deployment, logical
IoT-PML [89]	Physical, deployment, logical
Centurión et al. [90]	Physical, deployment, logical
COMFIT [91]	Physical, deployment, logical, behavior
HSML [92]	Physical, behavior, deployment
IoTDraw and SoaML4IoT [93,94]	Physical, deployment, logical
Alulema et al. [44]	Physical, deployment, logical
EL4IoT [95,96]	Physical, behavior, deployment, logical
Asici et al. [97]	Physical, deployment, logical
Sosa-Reyna et al. [98,99]	Physical, deployment, logical
Genesis [100,101]	Physical, deployment, logical
BRAIN-IoT [102]	Physical, logical, behavior, deployment
IADev [103]	Physical, deployment, logical
Iovino et al. [104]	Physical, logical, behavior, deployment
IoTSuite [105,106]	Physical, deployment, logical
Vitruvius [107]	Physical, deployment, logical
Midgar [108,109]	Physical, deployment, logical
Schachinger and Kastner [110]	Physical, deployment, logical
Karkouch et al. [111]	Physical, deployment, logical
IoTLink [37,112,113]	Physical, deployment, logical
SysML4IoT [114,115]	Physical, logical, behavior, deployment
FRASAD [116]	Physical, deployment, logical
Vorto [117]	Logical, behavior
DivEnact [118]	Physical, logical, behavior, deployment
ENACT DevOps [119]	Physical, behavior, logical, information, deployment
FogUML2Code [43]	Physical, deployment, logical
Betancourt et al. [120]	Physical, deployment, logical
MAF [121]	Physical, deployment, logical
Hassine et al. [122]	Physical, deployment, logical
SecKit [123]	Logical, behavior

The analysis results indicate that none of the modeling languages for IoT support all the viewpoints considered (i.e., logical, information, behavior, development, physical, deployment, concurrency, and operational viewpoints). However, the viewpoints such as information and concurrency that are not particularly supported by the languages are likely to be very useful to the implementers during IoT software development. Indeed, in many IoT problems, data in different formats need to flow between different IoT units. Moreover, IoT systems carry out data manipulations and data storage operations. So, we strongly believe that information viewpoint modeling is highly important for IoT software development. The IoT domain structure includes heterogeneous units with different and distributed functions. Simultaneous operation is very important because of this distributed unit layout. It is thought that this situation can be improved with the support of the concurrency viewpoint.

5.2.2. Extensibility

Extensibility is to do with the ability to extend a language with new language definition concepts, language features, and tools [131–133]. This includes support for new viewpoints, adding domain-specific features (e.g., identifying and analyzing nonfunctional features); expanding tool support with the necessary possibilities (e.g., formal analysis, simulation, and code generation); adding/removing new elements (e.g., complex connectors); or introducing a subnotation (e.g., a visual notation set).

Only three modeling languages for IoT (9%) support extensibility: (1) IoT-PML [89]; (2) IoTDraw and SoaML4IoT [93,94]; and (3) Vorto [117]. IoT-PML [89] supports extensibility with the addition of new modeling elements for different layers of the metamodel definition, and the new element is expected to follow the rules of that layer in which it is defined. IoTDraw and SoaML4IoT's [93,94] extensibility enables defining new properties for the existing modeling elements and defining new (executable) semantics for the existing modeling elements under some restrictions. In Vorto [117], the language definition cannot be extended. However, Vorto is the only language that provides support (SDKs) for introducing new code generators and transformation tools (i.e., the tools that can transform models in another modeling language into the model in Vorto).

5.2.3. Programming Framework

The programming framework includes support for IoT such as design patterns, architectural style, and generating library code from the model. All of the 32 languages analyzed offer an editor to specify software/hardware models textually/visually and check the models against the syntax and semantic rules of the languages. As shown in Table 5, the language editors also support the automatic code generation from model specifications. Figure 5 also shows the percentages of the modeling languages and their tools that support each programming language for code generation. Code generation in Java has the highest rate (34%), which is followed by C (21%), JavaScript (13%), C++ (11%), configuration files (XML, manifest, etc.) (9%), Obj-C (4%), nesC (4%), C# (2%), and Python (2%).

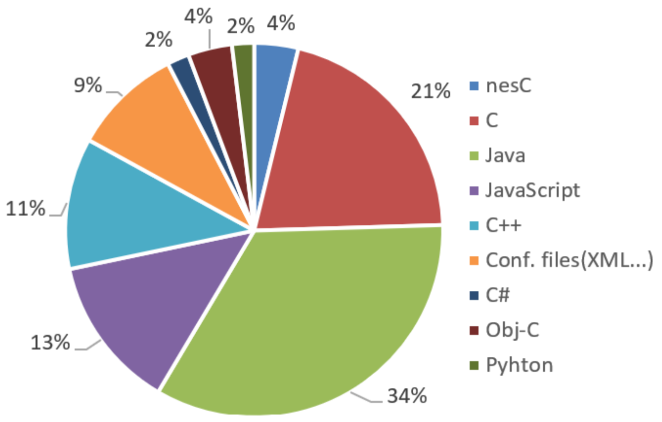

Figure 5. The percentages of the modeling languages that support different programming languages.

Table 5. The modeling languages for IoT that support automated code generation via their tools.

Language	Programming Languages
DSML4TinyOS [81–83]	nesC
ThingML [22,84–86]	C/C++, Java, JavaScript
UML4IoT [36,87,88]	C, Java
IoT-PML [89]	Java
Centurión et al. [90]	pp configuration file
COMFIT [91]	nesC, JavaScript
HSML [92]	JavaScript
IoTDraw and SoaML4IoT [93,94]	Java
Alulema et al. [44]	C, Java
EL4IoT [95,96]	nesC, XML config file
Asici et al. [97]	C, Java
Sosa-Reyna et al. [98,99]	nesC
Genesis [100,101]	C/C++, Java, JavaScript
BRAIN-IoT [102]	Java, Manifest script file
IADev [103]	Java, JavaScript
Iovino et al. [104]	Java
IoTSuite [105,106]	Java
Vitruvius [107]	JavaScript
Midgar [108,109]	C, Java, C#, Obj-C
Schachinger and Kastner [110]	Java
Karkouch et al. [111]	Java
IoTLink [37,112,113]	Java
SysML4IoT [114,115]	Java
FRASAD [116]	C/C++, Java
Vorto [117]	C/C++, Java, Python
DivEnact [118]	Configuration file
ENACT DevOps [119]	C/C++, Java, JavaScript
FogUML2Code [43]	C, Java
Betancourt et al. [120]	Configuration file
MAF [121]	Java, JSON, Obj-C
Hassine et al. [122]	Java
SecKit [123]	C/C++, Java

Some of the modeling languages for IoT support four different programming languages at the same time. These are (1) ThingML [22,84–86] with its support for C, C++, Java, and JavaScript; (2) Genesis [100,101] with its support for C, C++, Java, and JavaScript; (3) Midgar [108,109] with its support for C, Java, C#, and Obj-C; (4) Vorto [117] with its support for C, C++, Java, and Python; and (5) ENACT DevOps [119] with its support for C, C++, Java, and JavaScript.

Note that none of the modeling languages supports pattern-centered design (i.e., modeling with the reuse of existing patterns or modeling for defining reusable patterns).

5.3. Tool Support

5.3.1. Model Analysis

According to the analysis results, 22 modeling languages (69%) support the automatic analysis of models for completeness, consistency, correctness, and compatibility. Completeness relates to the level of detail contained in the characteristics of design decisions (e.g., system behavior and interaction) and is used to check whether the design decisions take into account all the requirements of the system. Consistency is about any conflicts between elements (e.g., components, interfaces, and connectors) specified in the model. This includes the consistency between the name of a service required and provided by the ports, connecting a required component port with a provided component port, the consistency between the required and provided behaviors of a service, and the consistency between the order of services required and the order of services provided. Correctness is taken into account to check whether the specified design decisions meet the desired system specifications (e.g., deadlock, race condition, or any user-defined properties). Finally, compatibility is about checking that software specifications conform to any architectural style or design guidelines.

As shown in Table 6, none of the modeling languages for IoT is supported with a tool that can analyze the models for all the analysis goals considered. The most preferred analysis goals are consistency (in 17 languages) (77%) and completeness (in 14 languages) (64%). Accuracy and compatibility targets are addressed by three (14%) languages each. There are two languages (Centurión et al. [90] and COMFIT [91]) that support completeness, consistency, and correctness at the same time, while only one language (ThingML [22,84–86]) supports completeness, consistency, and compatibility. Other languages support two or less features.

During the examinations on modeling languages for IoT, other requirements which these languages commonly support emerged. These are simulation, model checking, deadlock checking, and reachability. Simulation is to do with executing a software model to determine any wrong sequences of actions and wrong change of system state. Model checking (or theorem proof) is performed by model checking tools (or theorem provers) that rely on mathematical proofs and allow for comprehensive analysis of the models so as to prove the correctness of models for some properties. Indeed, model checkers can check each state a system can be in at any time for any features of interest. Note that although the simulation shows system execution in a single path of the system state space, model checkers check all system paths. Deadlock is a property used to check if any system with concurrently executing components has entered into a deadlock state, where each component gets blocked and expects a response from another component that will never happen. Reachability is another popular property for checking a system state that is never reached during system execution and reveals any part of behavior models that is not working as expected.

Table 7 shows the modeling languages that support each of these model analysis requirements discussed in the above paragraph. Model checking is the most popular analysis technique supported by ten languages (77%). Some languages use their own built-in model checkers, while others use model checkers such as nuXmv [134] (used by UML4IoT [37,87,88], BRAIN-IoT [102], and MAF [121]) and EMFeR [135] (used by IoTLink [37,112,113] and FogUML2Code [43]). Note that these languages basically provide translation of software features in the input language of model controllers. Additionally, nine out of thirteen languages (69%) were found to support deadlock detection via the model checkers they support. Five languages support simulation; most of them basically support model checkers with simulation capability (SPIN [136] and FSP [137]). Finally, reachability analysis is only considered by ENACT DevOps [119].

Table 6. The modeling languages for IoT that support the model analysis for the completeness, consistency, correctness, and compatibility properties.

Language	Completeness	Consistency	Correctness	Compatibility
DSML4TinyOS [81–83]	Yes	Yes	NA	NA
ThingML [22,84–86]	Yes	Yes	NA	Yes
UML4IoT [36,87,88]	NA	NA	NA	Yes
IoT-PML [89]	NA	NA	NA	Yes
Centurión et al. [90]	Yes	Yes	Yes	NA
COMFIT [91]	Yes	Yes	Yes	NA
HSML [92]	NA	Yes	NA	NA
IoTDraw and SoaML4IoT [93,94]	NA	NA	Yes	NA
Asici et al. [97]	Yes	Yes	NA	NA
Sosa-Reyna et al. [98,99]	NA	Yes	NA	NA
BRAIN-IoT [102]	Yes	Yes	NA	NA
IADev [103]	Yes	Yes	NA	NA
Iovino et al. [104]	Yes	NA	NA	NA
Karkouch et al. [111]	Yes	NA	NA	NA
IoTLink [37,112,113]	Yes	Yes	NA	NA
Vorto [117]	Yes	Yes	NA	NA
DivEnact [118]	NA	Yes	NA	NA
ENACT DevOps [119]	Yes	Yes	NA	NA
FogUML2Code [43]	Yes	Yes	NA	NA
Betancourt et al. [120]	NA	Yes	NA	NA
MAF [121]	Yes	Yes	NA	NA
SecKit [123]	NA	Yes	NA	NA
Total	14	17	3	3

Abbreviation: NA, not applicable.

5.3.2. Large-View Management

Large-view management is to do with managing large system models that are composed of many components which make the models difficult to understand and analyze. Therefore, different techniques such as structural composition (i.e., using composite components and connectors whose internal structures can be specified) and inheritance can be employed for minimizing the complexity by grouping the cohesive set of system components together.

According to the analysis results, 27 modeling languages for IoT (84%) were observed to address the large views. As can be seen in Table 8, the structural composition of component specifications (i.e., composite components) is the most supported requirement (66%). The structural compositions of both components and connectors are rarely supported (19%). Lastly, a few languages (16%) support the structural compositions of components and inheritance. Inheritance in this study is based on the inheritance principle of the object-oriented software engineering paradigm [138] and promotes the ability of extending a component specification with another component specification.

Table 7. The modeling languages for IoT that support the model analysis via some additional capabilities.

Language	Simulation	Model Checking	Deadlock	Reachability
DSML4TinyOS [81–83]	NA	Yes	Yes	NA
ThingML [22,84–86]	NA	Yes	NA	NA
UML4IoT [36,87,88]	NA	Yes	Yes	NA
COMFIT [91]	Yes	NA	NA	NA
IoTDraw and SoaML4IoT [93,94]	Yes	NA	NA	NA
Asici et al. [97]	NA	Yes	Yes	NA
Genesis [100,101]	Yes	NA	NA	NA
BRAIN-IoT [102]	NA	Yes	Yes	NA
IoTLink [37,112,113]	NA	Yes	Yes	NA
Vorto [117]	NA	Yes	Yes	NA
ENACT DevOps [119]	Yes	Yes	Yes	Yes
FogUML2Code [43]	NA	Yes	Yes	NA
MAF [121]	NA	Yes	Yes	NA
Total	5	10	9	1

Abbreviation: NA, not applicable.

5.3.3. Versioning

Versioning support is about maintaining and accessing different versions of software features. Only one modeling language for IoT, Vorto [117], supports versioning. This language mainly offers the use of repositories through modeling editors. Repositories can be used to store any supported architectural element (e.g., components and connectors), version these elements, access the versions needed, and reuse the versions to create different system configurations.

5.3.4. Knowledge Management

We consider knowledge management in terms of the availability of a website that can enable access to a set of artifacts including a language's (1) introduction, (2) tutorials, (3) toolset installation and download link, (4) user manual, (5) case studies, (6) publications, (7) mailing list, (8) membership, and (9) contact information.

The introduction support is to do with informing practitioners about the language. Tutorials refer to the usage guidelines of the language. The toolset requirement enables access to the language's tool itself. The user manual requirement is to do with the installation and usage information for practitioners. The case studies present the demonstrations of using the language and its toolset. Publications refer to the papers introducing the language features. The mailing list requirement is to do with any social platform through which practitioners exchange ideas about their questions on the language, toolset, and modeling. Membership means that there is a tool membership opportunity. Contact describes the existence of people who can be contacted for the language.

Table 9 gives the six modeling languages for IoT (19%) that have public websites. A careful examination of the websites of these languages identified a number of requirements that the websites (at least partially) support. These include tutorials, a toolset, a user manual, case studies, publications, a mailing list, membership, and contact info. Table 10 groups the modeling languages for IoT in terms of a set of knowledge management requirements. While some languages are not supported with websites, those languages still enable access to some of the information considered in our study through their publications and some repositories (e.g., GIT).

Table 8. The modeling languages for IoT that support large-view management.

Language	Large-View Management Techniques
DSML4TinyOS [81–83]	Structural composition (components)
ThingML [22,84–86]	Structural composition (components and connectors)
UML4IoT [36,87,88]	Structural composition (components)
IoT-PML [89]	Structural composition (components) and inheritance
Centurión et al. [90]	Structural composition (components)
HSML [92]	Structural composition (components)
IoTDraw and SoaML4IoT [93,94]	Structural composition (components) and inheritance
Alulema et al. [44]	Structural composition (components)
EL4IoT [95,96]	Structural composition (components)
Asici et al. [97]	Structural composition (components)
Sosa-Reyna et al. [98,99]	Structural composition (components)
Genesis [100,101]	Structural composition (components)
BRAIN-IoT [102]	Structural composition (components and connectors)
IADev [103]	Structural composition (components) and inheritance
IoTSuite [105,106]	Structural composition (components)
Vitruvius [107]	Structural composition (components)
Midgar [108,109]	Structural composition (components)
Schachinger and Kastner [110]	Structural composition (components)
IoTLink [37,112,113]	Structural composition (components)
SysML4IoT [114,115]	Structural composition (components and connectors)
FRASAD [116]	Structural composition (components)
Vorto [117]	Structural composition (components) and inheritance
DivEnact [118]	Structural composition (components and connectors)
ENACT DevOps [119]	Structural composition (components)
FogUML2Code [43]	Structural composition (components and connectors)
MAF [121]	Structural composition (components)
SecKit [123]	Structural composition (components) and inheritance

Table 9. The modeling languages for IoT that have accessible websites.

Language	Websites	Last Access Date
UML4IoT [36,87,88]	https://sites.google.com/site/uml4iot/	25 December 2022
BRAIN-IoT [102]	https://www.brain-iot.eu/	26 December 2022
IoTSuite [105,106]	https://bosch-iot-suite.com/	24 December 2022
IoTLink [37,112,113]	https://iotlink.gitlab.io/	25 December 2022
Vorto [117]	https://www.eclipse.org/vorto/	27 December 2022
ENACT DevOps [119]	https://www.enact-project.eu/	26 December 2022

Table 10. The modeling languages for IoT that support the knowledge management requirements.

Language	Introduction	Tutorials	Toolset	User Manual	Case Studies	Publication	Mailing List	Membership	Contact
DSML4TinyOS [81–83]	Yes	NA	Yes	NA	Yes	Yes	NA	NA	NA
ThingML [22,84–86]	Yes	Yes	Yes	Yes	Yes	Yes	NA	NA	Yes
UML4IoT [36,87,88]	Yes	Yes	Yes	Yes	Yes	Yes	NA	NA	Yes
IoT-PML [89]	Yes	Yes	Yes	Yes	Yes	Yes	NA	NA	NA
Centurión et al. [90]	Yes	NA	NA	NA	Yes	Yes	NA	NA	NA
COMFIT [91]	Yes	Yes	Yes	NA	Yes	Yes	NA	NA	NA
HSML [92]	Yes	NA	Yes	NA	Yes	Yes	NA	NA	NA
IoTDraw and SoaML4IoT [93,94]	Yes	Yes	Yes	Yes	Yes	Yes	NA	NA	Yes
Alulema et al. [44]	Yes	NA	Yes	NA	Yes	Yes	NA	NA	NA
EL4IoT [95,96]	Yes	NA	Yes	NA	Yes	Yes	NA	NA	NA
Asici et al. [97]	Yes	NA	Yes	NA	Yes	Yes	NA	NA	NA
Sosa-Reyna et al. [98,99]	Yes	NA	Yes	NA	Yes	Yes	NA	NA	NA
Genesis [100,101]	Yes	NA	Yes	NA	Yes	Yes	NA	NA	NA
BRAIN-IoT [102]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
IADev [103]	Yes	Yes	Yes	NA	Yes	Yes	NA	NA	NA
Iovino et al. [104]	Yes	NA	Yes	NA	Yes	Yes	NA	NA	NA
IoTSuite [105,106]	Yes	Yes	Yes	Yes	Yes	Yes	NA	NA	Yes
Vitruvius [107]	Yes	NA	Yes	NA	Yes	Yes	NA	NA	NA
Midgar [108,109]	Yes	NA	Yes	NA	Yes	Yes	NA	NA	NA
Schachinger and Kastner [110]	Yes	NA	Yes	NA	Yes	Yes	NA	NA	NA
Karkouch et al. [111]	Yes	NA	Yes	NA	Yes	Yes	NA	NA	NA
IoTLink [37,112,113]	Yes	NA	Yes	NA	Yes	Yes	NA	NA	NA
SysML4IoT [114,115]	Yes	NA	Yes	NA	Yes	Yes	NA	NA	NA
FRASAD [116]	Yes	NA	Yes	NA	Yes	Yes	NA	NA	NA
Vorto [117]	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
DivEnact [118]	Yes	NA	Yes	NA	Yes	Yes	NA	NA	NA
ENACT DevOps [119]	Yes	Yes	Yes	Yes	Yes	Yes	NA	NA	Yes
FogUML2Code [43]	Yes	NA	Yes	NA	Yes	Yes	NA	NA	NA
Betancourt et al. [120]	Yes	NA	Yes	NA	Yes	Yes	NA	NA	NA
MAF [121]	Yes	NA	Yes	NA	Yes	Yes	NA	NA	NA
Hassine et al. [122]	Yes	NA	Yes	NA	NA	Yes	NA	NA	NA
SecKit [123]	Yes	NA	Yes	NA	Yes	Yes	NA	NA	NA

Abbreviation: NA, not applicable.

The mailing list support and membership support are the least supported requirements among the languages analyzed. On the other hand, almost all languages have introduction artifacts, publications, toolsets, and case study information.

The two languages that support most of the knowledge management requirements are BRAIN-IoT [102] and Vorto [117]. Both languages come with introduction material, tutorials, a toolset, a user manual, case studies, publications, a mailing list, membership, and contact information.

The three modeling languages for IoT are limited by supporting only three knowledge management requirements. These languages and their supporting features are: (1) Centurión et al. [90]: introduction, case studies, and publications; (2) Sosa-Reyna et al. [98,99]: introduction, case studies, and publications; and (3) Hassine et al. [122]: introduction, toolset, and publications.

5.3.5. Software Architecture-Centric Design

For architecture-centric design, we analyzed the modeling languages and their tools for a set of features that we believe promote an architecture-centric design. These are (1) requirements modeling (e.g., use-case modeling); (2) traceability between requirements and architecture; (3) the automated analysis of architecture models for the requirement specifications; (4) the automated generation of code from the architecture models; and (5) reverse engineering (from code to architecture). Table 11 lists these features and the languages that support those features. Requirements modeling (34%) is the most supported feature among the languages. The support ratio for the automated analysis of the architecture for requirements is 9%. The support ratios for reverse engineering and the automated generation of code from the architecture are both 6%. None of the languages supports traceability between requirements and architecture.

Table 11. The modeling languages for IoT that support software architecture-centric design features.

Features	Languages
Requirements modeling	UML4IoT [36,87,88], IoT-PML [89], Centurión et al. [90], COMFIT [91], IoTDraw and SoaML4IoT [93,94], BRAIN-IoT [102], IADev [103], Karkouch et al. [111], SysML4IoT [114,115], FRASAD [116], SecKit [123]
The automated analysis of architecture for requirements	COMFIT [91], IADev [103], Karkouch et al. [111]
The automated generation of code from architecture	UML4IoT [36,87,88], COMFIT [91]
Reverse engineering	DSML4TinyOS [81–83], UML4IoT [36,87,88]

5.3.6. Collaboration

Teamwork should be an essential part of software development to develop high-quality software systems within expected time and budget constraints. Teamwork requires a group of people to work collaboratively on the same project, which reduces development time and improves product quality [139]. Collaboration within the human team can be synchronous or asynchronous. Synchronous collaboration requires users to simultaneously work on software features while they may be in physically different locations. In asynchronous collaboration, different users can work on specifications at different times that best fit their schedules. It was seen in this research that two languages (6%) provide both synchronous and asynchronous collaborations. These are Vorto [117] and ENACT DevOps [119].

6. Discussion

In Section 5, the analysis of 32 modeling languages for IoT was presented in terms of their support for three categories of requirements (i.e., language definition, language features, and tool support). Table 12 gives the overall analysis results for the IoT-based modeling languages. None of the IoT-based modeling languages supports all the requirements of language definition, language features, and tool support together.

We observed that the modeling languages for IoT provide different levels of support for the requirements in the different categories considered. Indeed, while one language supports a number of viewpoints for modeling, another one supports just two modeling viewpoints. Likewise, while one language supports various properties to be analyzed, another one just supports simulation. It should also be noted that such requirements as formal semantics, extensibility, and multi-user access (i.e., versioning and collaboration) are rarely supported. Vorto [117] can, for instance, be considered as supporting the greatest number of requirements while suffering from an informal semantics definition and a small number of modeling viewpoints supported (logical and behavior). IoTDraw and SoaML4IoT [93,94] support most requirements including formal semantics and extensibility but do not support versioning and collaboration features.

Table 12. The overall analysis results of the modeling languages for all the requirements considered.

Language Requirements:	**Language Definition**				**Language Features**					**Tool Support**				
	NonFunc Req	**NotSet**	**RY**	**ForSem**	**MulView**	**ExMec**	**ExtType**	**ProgFrame**	**Automated Analysis**	**LVM**	**Col**	**Ver**	**Knowledge Management**	**S-C**
DSML4TinyOS [81–83]	Per	GN	2018	IS	Phy, Dep, Log	NA	NA	ACG—nesC	MC, Dead—Comp, Cons	Yes	NA	NA	Int, Tool, CS, Pub	Yes
ThingML [22,84–86]	Per, Sc, Rel	GN	2016	IS	Phy, Log, Dep, Beh	NA	NA	ACG—C/C++, Java, JavaScript	MC—Comp, Cons, Compa,	Yes	NA	NA	Int, Tut, Tool, UM, CS, Pub, Con	Yes
UML4IoT [36,87,88]	Per, Rel, Sec, Time, RC	GN	2016	IS	Phy, Dep, Log	NA	NA	ACG—C, Java	MC, Dead—Compa	Yes	NA	NA	Int, Tut, Tool, UM, CS, Pub, Con	Yes
IoT-PML [89]	Gen	GN	2019	IS	Phy, Dep, Log	GCD	Syntax	ACG—Java	Compa	Yes	NA	NA	Int, Tut, Tool, UM, CS, Pub	Yes
Centurión et al. [90]	Sec, Sca	GN	2019	IS	Phy, Dep, Log	NA	NA	ACG—pp conf.	Comp, Cons, Corr	Yes	NA	NA	Int, CS, Pub	Yes
COMFIT [91]	Per, Rel, Sec, Por, RC	GN	2017	IS	Phy, Dep, Log, Beh	NA	NA	ACG—nesC, JavaScript	Sim—Comp, Cons, Corr	NA	NA	NA	Int, Tut, Tool, CS, Pub	Yes
HSML [92]	Per, RC	FSM and HTML	2019	IS	Phy, Dep, Beh	NA	NA	ACG—JavaScript	Cons	Yes	NA	NA	Int, Tool, CS, Pub	Yes
IoTDraw and SoaML4IoT [93,94]	Gen	GN	2020	fUML	Phy, Dep, Log	InExt	Syntax	ACG—Java	Sim—Corr	Yes	NA	NA	Int, Tut, Tool, UM, CS, Pub, Con	Yes
Alulema et al. [44]	Gen	GN	2017	IS	Phy, Dep, Log	NA	NA	ACG—C, Java	NA	Yes	NA	NA	Int, Tool, CS, Pub	Yes
EL4IoT [95,96]	Gen	GN	2017	IS	Phy, Dep, Beh, Log	NA	NA	ACG—nesC, XML	NA	Yes	NA	NA	Int, Tool, CS, Pub	Yes
Asici et al. [97]	Per	GN	2019	Petri net	Phy, Dep, Log	NA	NA	ACG—C, Java	MC, Dead—Comp, Cons	Yes	NA	NA	Int, Tool, CS, Pub	Yes
Sosa-Reyna et al. [98,99]	Gen	GN	2018	IS	Phy, Dep, Log	NA	NA	ACG—nesC	Cons	Yes	NA	NA	Int, CS, Pub	Yes
Genesis [100,101]	Gen	GN	2019	IS	Phy, Dep, Log	NA	NA	ACG—C/C++, Java, JavaScript	Sim	Yes	NA	NA	Int, Tool, CS, Pub	Yes
BRAIN-IoT [102]	Per, Sec, RC	GN	2019	IS	Phy, Dep, Log, Beh	NA	NA	ACG—Java, Man. Script	MC, Dead—Comp, Cons	Yes	NA	NA	Int, Tut, Tool, UM, CS, Pub, ML, Mem, Con	Yes
IADev [103]	Per, Rel, Sca	GN	2020	IS	Phy, Dep, Log	NA	NA	ACG—Java, JavaScript	Comp, Cons	Yes	NA	NA	Int, Tut, Tool, CS, Pub	Yes
Iovino et al. [104]	Sec, Ava	GN	2019	IS	Phy, Log, Dep, Beh	NA	NA	ACG—Java	Comp	NA	NA	NA	Int, Tool, CS, Pub	Yes
IoTSuite [105,106]	Sca	GN	2015	IS	Phy, Dep, Log	NA	NA	ACG—Java	NA	Yes	NA	NA	Int, Tut, Tool, UM, CS, Pub, Con	Yes

Table 12. *Cont.*

Language Requirements:	Language Definition				Language Features					Tool Support				
	NonFunc Req	NotSet	RY	ForSem	MulView	ExMec	ExtType	ProgFrame	Automated Analysis	LVM	Col	Ver	Knowledge Management	S-C
Vitruvius [107]	Gen	GN	2014	IS	Phy, Dep, Log	NA	NA	ACG—JavaScript	NA	Yes	NA	NA	Int, Tool, CS, Pub	Yes
Midgar [108,109]	Gen	GN	2014	IS	Phy, Dep, Log	NA	NA	ACG—C, Java, C#, Obj-C	NA	Yes	NA	NA	Int, Tool, CS, Pub	Yes
Schachinger and Kastner [110]	Gen	GN	2015	IS	Phy, Dep, Log	NA	NA	ACG—Java	NA	Yes	NA	NA	Int, Tool, CS, Pub	Yes
Karkouch et al. [111]	Gen	GN	2016	IS	Phy, Dep, Log	NA	NA	ACG—Java	Comp	NA	NA	NA	Int, Tool, CS, Pub	Yes
IoTLink [37,112,113]	Per	GN	2014	IS	Phy, Dep, Log	NA	NA	ACG—Java	MC, Dead—Comp, Cons	Yes	NA	NA	Int, Tool, CS, Pub	Yes
SysML4IoT [114,115]	Gen	GN	2016	IS	Phy, Log, Dep, Beh	NA	NA	ACG—Java	NA	Yes	NA	NA	Int, Tool, CS, Pub	Yes
FRASAD [116]	Por	GN	2015	IS	Phy, Dep, Log	NA	NA	ACG—C/C++, Java	NA	Yes	NA	NA	Int, Tool, CS, Pub	Yes
Vorto [117]	Gen	TN	2016	IS	Logical, Beh	JExt	NA	ACG—C/C++, Java, Python	MC, Dead—Comp, Cons	Yes	Yes	NSM	Int, Tut, Tool, UM, CS, Pub, ML, Mem, Con	Yes
DivEnact [118]	Rel	GN	2019	IS	Phy, Log, Dep, Beh	NA	NA	ACG—conf. file	Cons	Yes	NA	NA	Int, Tool, CS, Pub	Yes
ENACT DevOps [119]	Gen	GN	2018	IS	Phy, Dep, Beh, Log, Info	NA	NA	ACG—C/C++, Java, JavaScript	MC, Dead, Sim, Reac- Comp, Cons	Yes	Yes	NA	Int, Tut, Tool, UM, CS, Pub, Con	Yes
FogUML2Code [43]	Sec	GN	2019	IS	Phy, Dep, Log	NA	NA	ACG—C, Java	MC, Dead—Comp, Cons	Yes	NA	NA	Int, Tool, CS, Pub	Yes
Betancourt et al. [120]	Gen	GN	2020	IS	Phy, Dep, Log	NA	NA	ACG—conf. file	Cons	NA	NA	NA	Int, Tool, CS, Pub	Yes
MAF [121]	Gen	GN	2020	IS	Phy, Dep, Log	NA	NA	ACG—Java, JSON, Obj-C	MC, Dead—Comp, Cons	Yes	NA	NA	Int, Tool, CS, Pub	Yes
Hassine et al. [122]	Gen	GN	2017	IS	Phy, Dep, Log	NA	NA	ACG—Java	NA	NA	NA	NA	Int, Tool, Pub	Yes
SecKit [123]	Per, Sec	Tree N.	2014	LTL	Log, Beh	NA	NA	ACG—C/C++, Java	Cons	Yes	NA	NA	Int, Tool, CS, Pub	Yes

Abbreviations: ACG: automated code generation, Ava: availability, Beh: behavior, Col: collaboration, Compa: compatibility, Comp: completeness, Cons: consistency, Con: contact, Corr: correctness, CS: case studies, Dead: deadlock, Dep: deployment, ExMec: extension mechanism, ExtType: extension types, ForSem: formal semantics, GCD: generic concept definition, Gen: general, GN: graphical notation, InExt: (i) creating new attributes, (ii) specializing existing elements, (iii) implementing user-defined execution modules, Info: information, Int: introduction, IS: informal semantics, JExt: extension for reusing and specializing model elements such as Java, LTL: Linear Temporal Logic, Log: logical, Mem: membership, MulView: multiple viewpoints, MC: model checking, ML: mailing list, NA: not available, NonFunc Req: nonfunctional requirements, NotSet: notation sets, NSM: namespace management, RC: resource consumption, LVM: large-view management, RY: release year, Per: Performance, Phy: physical, Por: portability, ProgFrame: programming framework, Pub: publication, Reac: reachability, Rel: reliability, S-C: software architecture-centric design, Sca: scalability, Sc: schedulability, Sec: security, Sim: simulation, Time: timeliness, TN: textual notation, Tool: toolset, Tut: tutorials, UM: user Manuel, Ver: versioning.

The reminder of this section presents the implications and lessons learned for each requirement category considered in our study. The threats to the validity of the conducted study is also given in Section 6.4.

6.1. Language Definition

As discussed in Section 5.1.1, the support for the nonfunctional requirements is quite common among the modeling languages for IoT. The languages support the specifications of some quality properties through some notations. Note, however, that the support for the nonfunctional properties such as schedulability, timeliness, and availability, which are highly crucial for the IoT domain, is very limited. UML4IoT [36,87,88] and COMFIT [91] are the two languages that provide the widest range of nonfunctional property specifications. While many languages support the specifications of nonfunctional properties (especially the performance properties), none of them enables processing those specifications via some tool support. Considering the ever-increasing importance of quality checking in IoT systems (especially for time performance and availability), the lack of support for the nonfunctional property analysis could limit the use of the existing modeling languages for IoT. Thus, future modeling languages for IoT can address this issue and provide syntax and semantics support for the precise specifications of nonfunctional properties and their analysis via some tool support for the detection of design errors early on.

Another implication about the language definition is that the modeling languages for IoT lack support for the formal and precise semantic definitions—the only exceptions are IoTDraw and SoaML4IoT [93,94], Asici et al. [97], and SecKit [123]. The lack of such definitions could, however, lead to communication problems where the model specifications can be interpreted differently by different stakeholders, and thus wrong systems can be developed in the end. Another problem here is to do with reasoning about the design decisions at the modeling stage. Imprecise (i.e., ambiguous) models cannot be analyzed rigorously. Indeed, analyzing (and even proving) the behavior models via some formal verification tools (e.g., model checkers) requires translation from modeling language definitions into the model checking tools' input language definition, and that could be possible if the modeling languages were supported with precise definitions. It should also be noted that the model behavior analysis is very important for the IoT domain due to the heterogeneity and high complexity of IoT systems. So, we strongly believe that the new languages to be defined for the IoT domain should be supported with precise definitions for the enhanced communication and rigorous analysis of models without which the models cannot be very useful for the development of quality systems.

6.2. Language Features

As discussed in Section 5.2.1, the modeling languages for IoT support multiple-viewpoints modeling for the understandable and manageable specifications of design decisions. Indeed, the physical, deployment, and logical concerns are the most popular viewpoints. However, one important implication is that the information and concurrency modeling viewpoints are rarely supported by the IoT-based modeling languages. That is, making decisions about information-related issues (e.g., data flow, data structure, and data state changes) and concurrency-related issues (e.g., deadlock avoidance, synchronization, threads, and processes) are not within the scope of many languages. It should be noted that information and concurrency concerns are highly crucial for the development of IoT systems, and making early design decisions and reasoning about them are thus very important for IoT development. Therefore, language engineers and tool vendors can consider those features in the future during their language and tool development for enabling the early modeling of such decisions that can even be analyzed with some tool support.

Concerning the programming framework support, we learned that the IoT-based modeling languages do not support pattern-centric modeling, including the ability of defining patterns, reusing user-defined/pre-defined patterns in model specifications, and merging different models in the same model. Moreover, the languages do not support the use of any repositories that promote the reuse of models or model elements (e.g., components and connectors). However, without the use of patterns or the support for reusability in models, practitioners cannot always make the optimal design decisions as quickly as possible and therefore develop quality IoT systems within the expected time and budget.

The extensibility feature is only supported by IoT-PML [89], IoTDraw and SoaML4IoT [93,94], and Vorto [117]. IoT-PML and IoTDraw and SoaML4IoT [93,94] focus on extending the language definition (i.e., language syntax and semantics) under some restrictions that prevent violating the metamodel rules. Extending tools is only supported by Vorto, which provides the development libraries for developing and integrating code generators and transformation tools. Most languages cannot be extended at all, and practitioners have to use the existing notation set and tools as they are.

We strongly believe that for any language to be widely used in industry, it needs to support (i) multiple modeling viewpoints that enable addressing the principal concerns in that industry separately and thus more understandably, (ii) reusability in modeling via patterns and libraries for quick and quality modeling, and (iii) extensibility for extending the language and tools for the needs of specific problems encountered. Unfortunately, none of the existing IoT-based modeling languages seems to satisfy all those requirements, and we hope that new languages and tools to be developed in the future can address those requirements.

6.3. Tool Support

As discussed in Section 5.3.1, many of the modeling languages for IoT support the automated analysis of models. While some languages are supported with tools for analyzing models with regard to the four important analysis goals (i.e., completeness, consistency, correctness, and compatibility), some languages are supported with tools that enable model checking for deadlock detection. Note, however, that while checking model consistency and model completeness are supported by many language tools, checking model compatibility for, e.g., architectural styles, is rarely supported. ThingML [22,84–86], UML4IoT [36,87,88], and IoT-PML [89] are the only languages that support compatibility analysis. A few language tools support model simulation too. However, those that support model simulation ignore model checking (i.e., formal verification)—the only exception here is ENACT DevOps [119].

Another interesting implication about tool support is that all the modeling languages for IoT are supported with tools for automated code generation. Java is the most popular programming language, followed by C and JavaScript. Moreover, some language tools support multiple programming languages for code generation. Indeed, ThingML [22,84–86], Genesis [100,101], and ENACT DevOps [119] support C, C++, Java, and JavaScript at the same time.

Large-view management is also supported by most of the modeling languages for IoT. While all of those languages enable the structural composition of components, IoT-PML [89], IoTDraw and SoaML4IoT [93,94], IADev [103], SecKit [123], and Vorto [117] further support the inheritance capability for the modeling elements where the modeling elements may be specified by specializing other existing elements.

The IoT-based modeling languages ignore model versioning and collaborative modeling for the multi-user access support. The only exceptions here are Vorto [117] and ENACT DevOps [119]. The rest of the languages do not enable (i) versioning models, (ii) comparing model versions, and (iii) multiple users to access the model versions at the same time. However, without versioning and multi-user access, modeling tools may not be applicable to the industrial cases where a single solution needs to be revised in several versions by different stakeholders collaboratively.

The architecture-centric design is another interesting requirement for tool support. While all the languages support the specifications of system architectures for IoT, not all of them enable the specifications of requirements and linking requirements with the architecture models and reasoning about the relationship here. Indeed, some languages support the requirements specifications too, but none of those languages is supported with the tools for establishing the traceability between requirements and architecture and making the necessary checks.

Knowledge management is supported by the languages to some extent. However, while it is possible to obtain different types of knowledge about the languages and tools (e.g., tutorials, case studies, publications, user manuals, toolsets, contact info, etc.), most languages are not supported with proper webpages where all the information can be accessed easily from a single resource.

Concerning the tool support of the modeling languages analyzed, Vorto [117] and ENACT DevOps [119] are the two that provide stronger tool support in terms of the requirements considered. ENACT DevOps [119] has a broad automated analysis support. Both Vorto [117] and ENACT DevOps [119] also support some collaboration features. Note, however, that ENACT DevOps [119] does not provide versioning support. Lastly, both languages offer extensive knowledge management via proper webpages.

6.4. Threats to Validity

As is the case in any analysis study, there may be some internal, external, construct, and conclusion threats to the validity of the findings in this study.

Internal validity is concerned with the causal relationships between the results of the analysis and any independent variable (i.e., cause) that leads to the results [140]. The goal here is to ensure that the analysis results are derived from the language requirements considered, and that unknown independent variables do not affect the results.

In this study, nonprobabilistic sampling was used, and the modeling languages for IoT to be analyzed were chosen non-randomly. That is, the languages were searched on the Internet and many different academic databases. In addition, the languages were analyzed for three groups of requirements: language definition, language features, and tool support. The requirements here are based on the research by Lago et al. [44]. Note that some requirements that are not precise enough on their own were broken down into sub-requirements (see Figure 2). In this way, we aimed to obtain more precise analysis results. The parsing here is performed using definitions from Lago et al. [45] and definitions of concepts from the seminal book by Taylor et al. [77].

The analysis of the languages for the specified requirements was performed by two researchers experienced in IoT software. Thus, we aimed to prevent instrumentation biases that may occur in case of a change in the researchers collecting and analyzing the data, and this change affects the analysis results due to the changing profiles of the researchers. As discussed in Section 4, the authors systematically conducted the analysis in several iterations. Initially, supporting materials for each language were reviewed, including publications, tutorials, and tools, to gain some familiarity with the languages. Then, in another iteration, the languages' supporting materials were reviewed to determine the level of support for those requirements. This process was repeated in exactly the same way to detect any inconsistencies.

External validity threats concern the generalizability of the analysis results, that is, the degree to which the examined studies are representative of the reviewed topic [140]. The set of modeling languages analyzed in our study may not be representative of the entire set of all existing studies on the modeling languages for IoT applications. However, this threat was mitigated by an extensive search of the related documents using popular search engines and academic publication repositories as discussed in Section 4. These documents consist of scientific papers, technical reports, theses, and user manuals. Further, the languages' websites were visited, and available modeling tools supporting these languages were carefully examined. Moreover, all collected materials were analyzed by all authors separately using a set of previously determined requirements. Only studies in English were included. Papers written in other languages concerning the same topic may exist. However, this threat is considered as having minimal effect.

Construct validity relates to how well an analysis helps in achieving the research objective. Our goal was to analyze existing modeling languages for IoT to see to what extent these languages are useful in IoT application development. For this purpose, we used three groups of language requirements which are well known and were previously used in different analysis and/or evaluation studies in various software engineering domains. As also discussed in Section 4, the analyzed data were compared, existing inconsistencies were determined, and these parts were re-analyzed together with all authors until they reached a consensus on them. This method also contributed to minimizing the risk to the construct validity of the conducted research. Additionally, we needed to ensure that all relevant studies on the selected topic were found adequately. Both MDE and IoT are well-established concepts, and thus these terms were good enough to be used as keywords. Therefore, to mitigate this risk, the search of the existing studies through several iterations was formed, and adequate coverage of the literature was achieved. It is worth noting that we excluded the previous surveys and reviews during these iterations as is usual in many systematic reviews. Although that may have caused us to skip some relevant primary studies discussed in these secondary studies, we determined that the wide coverage of our paper identification and screening processes already enabled us to consider all relevant primary studies also discussed in these previous secondary studies.

To alleviate the conclusion validity threat, the research methodology of this study was designed and validated carefully to minimize the risk of excluding relevant studies. Benefiting from our previous experience of conducting other surveys and SLRs (e.g., [23,68,141]), the search methodology for the IoT modeling languages was formalized and applied in a way that only a very small number of relevant studies could be missed, and a manageable quantity of irrelevant studies could be included. Finally, the analysis of the languages led to many interesting findings for each set of requirements that are expected to be very useful to IoT developers.

7. Conclusions and Future Work

The aim of this study was to analyze existing modeling languages for IoT to understand their support for requirements that are important for practitioners. We considered 32 different languages and analyzed each language for three sets of requirements, i.e., language definition, language features, and tool support. The language definition requirements are concerned with the (1) nonfunctional features, (2) notation set type (i.e., textual, visual, and hybrid), and (3) formal semantics. Language features are concerned with the (1) multiple viewpoints, (2) extensibility, and (3) programming framework. Finally, tool support is concerned with (1) automated analysis, (2) large-view management, (3) collaboration support, (4) versioning support, (5) knowledge management support, and (6) software architecture-centric design.

Some of the key findings of our analysis include: (1) among many quality properties, performance is the most popular one, supported by 28% of the languages; (2) almost all the languages provide a visual notation set exclusively, while 6% provide both textual and visual notation sets; (3) most of the languages (88%) lack formally precise semantic definitions; (4) most languages (94%) support the physical, deployment, and logical modeling viewpoints, while the behavior, logical, and information modeling viewpoints are rarely supported; (5) almost none of the languages support the extensibility of language syntax, semantics, and tool support; (6) Java (34%) and C (21%) are the most preferred programming languages in transforming models into code; (7) among the analysis properties considered, consistency (77%) and completeness (64%) are the most supported properties for automated checking; and (8) most of the languages (81%) are not supported with any websites for informing practitioners about case studies, source code, tools, tutorials, etc. Our results can serve as a guide when using or developing modeling languages for IoT. Users can consider these comparisons to select the most appropriate modeling language for their IoT system development purposes, while language developers, academic groups, and vendors will benefit from the analysis results in improving modeling features of in-use or future languages and tools.

In the future, we will validate the analysis results through a set of companies that produce IoT-based solutions. We will organize a series of interviews with the practitioners so as to understand their thoughts about the analysis results discussed in this paper, the properties that interest them in an IoT-based language, which of the IoT-based languages analyzed they prefer to use, the reasons behind their choice(s), and their suggestions for further analytical studies on these languages. Moreover, we also plan to design and execute a practitioner survey by taking into consideration the analysis results discussed in this paper. In this survey study, we will try to understand the IoT practitioners' perspectives on modeling, modeling languages, their expectations, and any difficulties faced during the IoT system development. Another future study could be on designing and developing a new domain-specific language for IoT where such crucial properties as extensibility and hybrid notation set that are ignored by most of the existing languages can be addressed.

Author Contributions: Investigation, S.A.; methodology, S.A., M.O. and G.K.; software, S.A., supervision, G.K.; validation, M.O. and G.K.; visualization, S.A.; writing—original draft, S.A., M.O. and G.K.; writing—review and editing, G.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Rayes, A.; Salam, S. *Internet of Things from Hype to Reality: The Road to Digitization*, 1st ed.; Springer: Cham, Switzerland, 2019; pp. 2–4.
2. Datta, S.K.; Costa, R.P.F.D.; Härri, J.; Bonnet, C. Integrating connected vehicles in Internet of Things ecosystems: Challenges and solutions. In Proceedings of the 17th IEEE International Symposium on A World of Wireless, Mobile and Multimedia Networks, Coimbra, Portugal, 21–24 June 2016.
3. Bauer, M.; Bui, N.; Giacomin, P.; Gruschka, N.; Haller, S.; Ho, E.; Kernchen, R.; Lischka, M.; Loof, J.D.; Magerkurth, C.; et al. Internet-of-Things Architecture IoT-A Project Deliverable D1.2—Initial Architectural Reference Model for IoT. Available online: https://cocoa.ethz.ch/downloads/2014/01/1360_D1%202_Initial_architectural_reference_model_for_IoT.pdf (accessed on 27 December 2022).
4. Ciccozzi, F.; Crnkovic, I.; Ruscio, D.D.; Malavolta, I.; Pelliccione, P.; Spalazzese, R. Model-Driven Engineering for MissionCritical IoT Systems. *IEEE Softw.* **2017**, *34*, 46–53. [CrossRef]
5. Udoh, I.S.; Kotonya, G. Developing IoT applications: Challenges and frameworks. *IET Cyber Phys. Syst. Theory Appl.* **2018**, *3*, 65–72. [CrossRef]
6. Noura, M.; Atiquzzaman, M.; Gaedke, M. Interoperability in Internet of Things: Taxonomies and open challenges. *Mob. Netw. Appl.* **2019**, *24*, 796–809. [CrossRef]
7. Berrouyne, I.; Adda, M.; Mottu, J.M.; Tisi, M. A Model-Driven Methodology to Accelerate Software Engineering in the Internet of Things. *IEEE Internet Things J.* **2022**, *9*, 19757–19772. [CrossRef]
8. Grace, P.; Pickering, B.; Surridge, M. Model-driven interoperability: Engineering heterogeneous IoT systems. *Ann. Telecommun. Telecommun.* **2015**, *71*, 141–150. [CrossRef]
9. Siegel, J.E.; Kumar, S.; Sarma, S.E. The Future Internet of Things: Secure, Efficient, and Model-Based. *IEEE Internet Things J.* **2018**, *5*, 2386–2398. [CrossRef]
10. Larrucea, X.; Combelles, A.; Favaro, J.; Taneja, K. Software engineering for the internet of things. *IEEE Softw.* **2017**, *34*, 24–28. [CrossRef]
11. Durmaz, C.; Yildirim, K.S.; Kardas, G. Virtualizing intermittent computing. *IEEE Internet of Things J.* **2022**, *9*, 20869–20878. [CrossRef]
12. Amin, F.; Abbasi, R.; Mateen, A.; Ali Abid, M.; Khan, S. A Step toward Next-Generation Advancements in the Internet of Things Technologies. *Sensors* **2022**, *22*, 8072. [CrossRef] [PubMed]
13. Amin, F.; Majeed, A.; Mateen, A.; Abbasi, R.; Hwang, S.O. A Systematic Survey on the Recent Advancements in the Social Internet of Things. *IEEE Access* **2022**, *10*, 63867–63884. [CrossRef]
14. Whittle, J.; Hutchinson, J.; Rouncefield, M. The state of practice in model-driven engineering. *IEEE Softw.* **2014**, *31*, 79–85. [CrossRef]

15. Brambilla, M.; Cabot, J.; Wimmer, M. *Model-Driven Software Engineering in Practice*, 2nd ed.; Springer: Berlin/Heidelberg, Germany, 2017; pp. 63–73.
16. Lelandais, B.; Oudot, M.P.; Combemale, B. Applying model-driven engineering to high-performance computing: Experience report, lessons learned, and remaining challenges. *J. Comput. Lang.* **2019**, *55*, 100919. [CrossRef]
17. Selic, B. The pragmatics of model-driven development. *IEEE Softw.* **2003**, *20*, 19–25. [CrossRef]
18. Seidewitz, E. What models mean. *IEEE Softw.* **2003**, *20*, 26–32. [CrossRef]
19. Kühne, T. Matters of (meta-)modeling. *Softw. Syst. Model.* **2006**, *5*, 369–385. [CrossRef]
20. Ozkaya, M. Are the UML modelling tools powerful enough for practitioners? A literature review. *IET Softw.* **2019**, *13*, 338–354. [CrossRef]
21. Strembeck, M.; Zdun, U. An approach for the systematic development of domain-specific languages. *Softw. Pract. Exp.* **2009**, *39*, 1253–1292. [CrossRef]
22. Morin, B.; Harrand, N.; Fleurey, F. Model-based software engineering to tame the iot jungle. *IEEE Softw.* **2017**, *34*, 30–36. [CrossRef]
23. Mohamed, M.A.; Challenger, M.; Kardas, G. Applications of model-driven engineering in cyber-physical systems: A systematic mapping study. *J. Comput. Lang.* **2020**, *59*, 100972. [CrossRef]
24. Rumbaugh, J.E.; Jacobson, I.; Booch, G. *The Unified Modeling Language Reference Manual*; Addison-Wesley-Longman: Boston, MA, USA, 1999.
25. SysML Open-Source Project. Available online: https://sysml.org/ (accessed on 27 October 2022).
26. Chinosi, M.; Trombetta, A. BPMN: An introduction to the standard. *Comput. Stand. Interfaces* **2012**, *34*, 124–134. [CrossRef]
27. The ArchiMate, Enterprise Architecture Modeling Language. Available online: https://www.opengroup.org/archimate-forum/archimate-overview (accessed on 27 October 2022).
28. Feiler, P.; Gluch, D.; Hudak, J. *The Architecture Analysis & Design Language (AADL): An Introduction*; Technical Note CMU/SEI-2006-TN-011; Software Engineering Institute, Carnegie Mellon University: Pittsburgh, PA, USA, 2006.
29. Zhao, J.J.; Ding, J.W. Modelica and Its Mechanism of Multi-domain Unified Modeling and Simulation. *J. Syst. Simul.* **2006**, *18*, 570–573.
30. Ommering, R.V.; Linden, F.V.D.; Kramer, J.; Magee, J. The Koala component model for consumer electronics software. *Computer* **2000**, *33*, 78–85. [CrossRef]
31. Eclipse Modelling Framework. Available online: https://www.eclipse.org/modeling/emf/ (accessed on 27 October 2022).
32. MetaEdit+ Domain-Specific Modeling (DSM) Environment. Available online: http://www.metacase.com/products.html (accessed on 27 October 2022).
33. Motta, R.C.; Oliveira, K.M.D.; Travassos, G.H. On challenges in engineering IoT software systems. In Proceedings of the XXXII Brazilian Symposium on Software Engineering, Sao Carlos, Brazil, 17–21 September 2018.
34. Sarhan, Q.I. Internet of Things: A Survey of Challenges and Issues. *Int. J. Internet Things Cyber-Assurance* **2018**, *1*, 40–75. [CrossRef]
35. Kölsch, J.; Post, S.; Zivkovic, C.; Ratzke, A.; Grimm, C. Model-based development of smart home scenarios for IoT simulation. In Proceedings of the 8th Workshop on Modeling and Simulation of Cyber-Physical Energy Systems, Sydney, NSW, Australia, 21 April 2020.
36. Thramboulidis, K.; Christoulakis, F. UML4IoT—A UML profile to exploit IoT in cyber-physical manufacturing systems. *Comput. Ind.* **2016**, *82*, 259–272. [CrossRef]
37. Khaleel, H.; Conzon, D.; Kasinathan, P.; Brizzi, P.; Pastrone, C.; Pramudianto, F.; Eisenhauer, M.; Cultrona, P.A.; Rusinà, F.; Lukáč, G.; et al. Heterogeneous Applications, Tools, and Methodologies in the Car Manufacturing Industry Through an IoT Approach. *IEEE Syst. J.* **2017**, *11*, 1412–1423. [CrossRef]
38. Mezghani, E.; Exposito, E.; Drira, K. A Model-Driven Methodology for the Design of Autonomic and Cognitive IoT-Based Systems: Application to Healthcare. *IEEE Trans. Emerg. Top. Comput. Intell.* **2017**, *1*, 224–234. [CrossRef]
39. Hause, M.; Hummell, J.; Grelier, F. MBSE Driven IoT for Smarter Cities. In Proceedings of the 13th Annual Conference on System of Systems Engineering, Paris, France, 19–22 June 2018.
40. García-López, A.; Burgueño, L.; Vallecillo, A. Static Analysis of Complex Event Processing Programs. In Proceedings of the MoDELS, Copenhagen, Denmark, 14–19 October 2018.
41. Lingen, F.V.; Yannuzzi, M.; Jain, A.; Irons-Mclean, R.; Lluch, O.; Carrera, D.; Pérez, J.L.; Gutierrez, A.; Montero, D.; Martí, J.; et al. The Unavoidable Convergence of NFV, 5G, and Fog: A Model-Driven Approach to Bridge Cloud and Edge. *IEEE Commun. Mag.* **2017**, *55*, 28–35. [CrossRef]
42. Sharaf, M.; Abusair, M.; Muccini, H.; Eleiwi, R.; Shana'a, Y.; Saleh, I. Generating heterogeneous codes for IoT systems. MODELS'19. In Proceedings of the 22nd International Conference on Model Driven Engineering Languages and Systems, Munich, Germany, 15–20 September 2019.
43. Pusztai, T.W.; Tsigkanos, C.; Dustdar, S. Engineering Heterogeneous Internet of Things Applications: From Models to Code. In Proceedings of the IEEE 5th International Conference on Collaboration and Internet Computing, Los Angeles, CA, USA, 12–14 December 2019.
44. Alulema, D.; Iribarne, L.; Criado, J. A DSL for the Development of Heterogeneous Applications. In Proceedings of the 5th International Conference on Future Internet of Things and Cloud Workshops, Prague, Czech Republic, 21–23 August 2017.

45. Lago, P.; Malavolta, I.; Muccini, H.; Pelliccione, P.; Tang, A. The road ahead for architectural languages. *IEEE Softw.* **2015**, *32*, 98–105. [CrossRef]
46. Ozkaya, M. The analysis of architectural languages for the needs of practitioners. *Softw. Pract. Exp.* **2018**, *48*, 985–1018. [CrossRef]
47. Medvidovic, N.; Taylor, R.N. A classification comparison framework for software architecture description languages. *IEEE Trans. Softw. Eng.* **2000**, *26*, 70–93. [CrossRef]
48. Clements, P.C. A survey of architecture description languages. In Proceedings of the 8th International Workshop on Software Specification and Design, Washington, DC, USA, 22–23 March 1996.
49. Medvidovic, N.; Dashofy, E.M.; Taylor, R.N. Moving architectural description from under the technology lamppost. *Inf. Softw. Technol.* **2007**, *49*, 12–31. [CrossRef]
50. Vestal, S.A. *Cursory Overview and Comparison of Four Architecture Description Languages*; Technical Report; Honeywell Technology Center: Washington, DC, USA, 1993.
51. Hilliard, R.; Rice, T. Expressiveness in architecture description languages. In Proceedings of the Third International Workshop on Software architecture, New York, NY, USA, 1 November 1998.
52. Ali, N.; Solís, C.; Ramos, I. Comparing architecture description languages for mobile software systems. In Proceedings of the 1st International Workshop on Software Architectures and Mobility, New York, NY, USA, 10 May 2008.
53. Mckenzie, F.D.; Petty, M.D.; Xu, Q. Usefulness of software architecture description languages for modeling and analysis of federates and federation architectures. *Simulation* **2004**, *80*, 559–576. [CrossRef]
54. Qin, W.; Malik, S. Architecture description languages for retargetable compilation. In *The Compiler Design Handbook*, 1st ed.; Srikant, Y.N., Shankar, P., Eds.; CRC Press: Boca Raton, FL, USA, 2002; Volume 14, pp. 534–564.
55. Mishra, P.; Dutt, N. Architecture description languages for programmable embedded systems. *IEE Proc. Comput. Digit. Tech.* **2005**, *152*, 285–297. [CrossRef]
56. Ozkaya, M.; Kloukinas, C. Are we there yet? Analyzing architecture description languages for formal analysis, usability, and realizability. In Proceedings of the 39th Euromicro Conference on Software Engineering and Advanced Applications, Santander, Spain, 4–6 September 2013.
57. Balaban, M.; Bennett, P.; Doan, K.H.; Georg, G.; Gogolla, M.; Khitron, I.; Kifer, M. A comparison of textual modeling languages: OCL, alloy, FOML. In Proceedings of the 16th International Workshop on OCL and Textual Modeling co-located with 19th International Conference on Model Driven Engineering Languages and Systems (MODELS 2016), Saint-Malo, France, 2 October 2016.
58. Bergmayr, A.; Breitenbücher, U.; Ferry, N.; Rossini, A.; Solberg, A.; Wimmer, M.; Kappel, G.; Leymann, F. A systematic review of cloud modeling languages. *ACM Comput. Surv.* **2018**, *51*, 1–38. [CrossRef]
59. Lelovic, L.; Mathews, M.; Elsayed, A.; Cerny, T.; Frajtak, K.; Tisnovsky, P.; Taibi, D. Architectural languages in the microservice era: A systematic mapping study. In Proceedings of the RACS'22: The Conference on Research in Adaptive and Convergent Systems, Virtual, 3–6 October 2022.
60. Alidra, A.; Bruneliere, H.; Ledoux, T. A feature-based survey of Fog modeling languages. *Future Gener. Comput. Syst.* **2023**, *138*, 104–119. [CrossRef]
61. Silva, E.A.; Valentin, E.; Carvalho, J.R.H.; Barreto, R.S. A survey of Model Driven Engineering in robotics. *J. Comput. Lang.* **2021**, *62*, 101021. [CrossRef]
62. Rashid, M.; Anwar, M.W.; Khan, A.M. Toward the tools selection in model based system engineering for embedded systems—A systematic literature review. *J. Syst. Softw.* **2015**, *106*, 150163. [CrossRef]
63. Liebel, G.; Marko, N.; Tichy, M.; Leitner, A.; Hansson, J. Model-based engineering in the embedded systems domain: An industrial survey on the state-of-practice. *Softw. Syst. Model.* **2018**, *17*, 91–113. [CrossRef]
64. Coronado, E.; Mastrogiovanni, F.; Indurkhya, B.; Venture, G. Visual Programming Environments for End-User Development of intelligent and social robots, a systematic review. *J. Comput. Lang.* **2020**, *58*, 100970. [CrossRef]
65. Geismann, J.; Bodden, E. A systematic literature review of model-driven security engineering for cyber-physical systems. *J. Syst. Softw.* **2020**, *169*, 110697. [CrossRef]
66. Liu, B.; Zhang, Y.R.; Cao, X.L.; Liu, Y.; Gu, B.; Wang, T.X. A survey of model-driven techniques and tools for cyber-physical systems. *Front. Inf. Technol. Electron. Eng.* **2020**, *21*, 15671590. [CrossRef]
67. Barišić, A.; Ruchkin, I.; Savić, D.; Mohamed, M.A.; Al-Ali, R.; Li, L.W.; Mkaouar, H.; Eslampanah, R.; Challenger, M.; Blouin, D.; et al. Multi-paradigm modeling for cyber–physical systems: A systematic mapping review. *J. Syst. Softw.* **2022**, *183*, 111081. [CrossRef]
68. Mohamed, M.A.; Kardas, G.; Challenger, M. Model-driven engineering tools and languages for cyber-physical systems—A systematic literature review. *IEEE Access* **2021**, *9*, 48605–48630. [CrossRef]
69. Li, L.; Aslam, S.; Wileman, A.; Perinpanayagam, S. Digital Twin in Aerospace Industry: A Gentle Introduction. *IEEE Access* **2021**, *10*, 9543–9562. [CrossRef]
70. Fortino, G.; Savaglio, C.; Spezzano, G.; Zhou, M. Internet of Things as System of Systems: A Review of Methodologies, Frameworks, Platforms, and Tools. *IEEE Trans. Syst. Man Cybern. Syst.* **2021**, *51*, 223–236. [CrossRef]
71. Silva, A.C.F.; Hirmer, P. Models for Internet of Things Environments—A Survey. *Information* **2020**, *11*, 487. [CrossRef]

72. Ihirwe, F.; Indamutsa, A.; Ruscio, D.D.; Mazzini, S.; Pierantonio, A. Cloud-based modeling in IoT domain: A survey, open challenges and opportunities. In Proceedings of the ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), Fukuoka, Japan, 10–15 October 2021.
73. Malavolta, I.; Lago, P.; Muccini, H.; Pelliccione, P.; Tang, A. What industry needs from architectural languages: A survey. *IEEE Trans. Softw. Eng.* **2012**, *39*, 869–891. [CrossRef]
74. Milner, R.; Parrow, J.; Walker, D. Calculus of mobile processes, I. *Inf. Comput.* **1992**, *100*, 1–40. [CrossRef]
75. Hoare, C.A.R. Communicating sequential processes. *Commun. ACM* **1978**, *21*, 666–677. [CrossRef]
76. Kruchten, P. The 4 + 1 view model of architecture. *IEEE Softw.* **1995**, *12*, 42–50. [CrossRef]
77. Taylor, R.N.; Medvidovic, N.; Dashofy, E.M. *Software Architecture: Foundations, Theory, and Practice*; Wiley: Hoboken, NJ, USA, 2010.
78. Rozanski, N.; Woods, E. *Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives*; Addison-Wesley Professional: Upper Saddle River, NJ, USA, 2005.
79. Di Ruscio, D.; Malavolta, I.; Muccini, H.; Pelliccione, P.; Pierantonio, A. Developing next generation ADLs through MDE techniques. In Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering, New York, NY, USA, 1–8 May 2010.
80. Page, M.J.; McKenzie, J.E.; Bossuyt, P.M.; Hoffmann, T.C.; Mulrow, C.D.; Shamseer, L.; Tetzlaff, J.M.; Akl, E.A.; Brennan, S.E.; Chou, R.; et al. The PRISMA 2020 statement: An updated guideline for reporting systematic reviews. *BMJ* **2021**, *372*, n71. [CrossRef]
81. Marah, H.M.; Eslampanah, R.; Challenger, M. DSML4TinyOS: Code Generation for Wireless Devices. In Proceedings of the ACM/IEEE 21st International Conference on Model Driven Engineering Languages and Systems (MODELS Workshop), Copenhagen, Denmark, 14–19 October 2018.
82. Marah, H.M.; Kardas, G.; Challenger, M. RE4TinyOS: A Reverse Engineering Methodology for the MDE of TinyOS Applications. In Proceedings of the Federated Conference on Computer Science and Information Systems, Online, 2–5 September 2021.
83. Marah, H.M.; Kardas, G.; Challenger, M. Model-driven round-trip engineering for TinyOS-based WSN applications. *J. Comput. Lang.* **2021**, *65*, 101051. [CrossRef]
84. Harrand, N.; Fleurey, F.; Morin, B.; Husa, K.E. Thingml: A language and code generation framework for heterogeneous targets. In Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering, Languages and Systems, Saint-malo, France, 2–7 October 2016.
85. Moin, A.; Rössler, S.; Sayih, M.; Günnemann, S. From Things' Modeling Language (ThingML) to Things' Machine Learning (ThingML2). In Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings 2020, Online, 16–23 October 2020.
86. Vasilevskiy, A.; Morin, B.; Haugen, Ø.; Evensen, P. Agile development of home automation system with ThingML. In Proceedings of the 2016 IEEE 14th International Conference on Industrial Informatics (INDIN), Poitiers, France, 19–21 July 2016.
87. Thramboulidis, K.; Foradis, T. From Mechatronic Components to Industrial Automation Things: An IoT Model for Cyber-Physical Manufacturing Systems. *J. Softw. Eng. Appl.* **2017**, *10*, 734–753.
88. Christoulakis, F.; Thramboulidis, K. IoT-based integration of IEC 61131 industrial automation systems: The case of UML4IoT. In Proceedings of the IEEE 25th International Symposium on Industrial Electronics (ISIE), Santa Clara, CA, USA, 8–10 June 2016; pp. 322–327.
89. Kühlwein, A.; Paule, A.; Hielscher, L.; Rosenstiel, W.; Bringmann, O. Firmware Synthesis for Ultra-Thin IoT Devices Based on Model Integration. In Proceedings of the ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), Munich, Germany, 15–20 September 2019.
90. Centurión, M.; Kotvinsky, M.; Calegari, D.; Delgado, A. A Model-Driven Approach for System Administration. In Proceedings of the 3th International Workshop on Model-Driven Engineering for the Internet-of-Things, Munich, Germany, 15–17 September 2019.
91. Farias, C.M.; Brito, I.C.; Pirmez, L.; Delicato, F.C.; Pires, P.F.; Rodrigues, T.C.; Santos, I.L.; Carmo, L.F.R.C.; Batista, T. COMFIT: A development environment for the Internet of Things. *Future Gener. Comput. Syst.* **2017**, *75*, 128–144. [CrossRef]
92. Xiao, R.; Wu, Z.; Wang, D. A Finite-State-Machine model driven service composition architecture for internet of things rapid prototyping. *Future Gener. Comput. Syst.* **2019**, *99*, 473–488. [CrossRef]
93. Costa, B.; Pires, P.F.; Delicato, F.C. Towards the adoption of OMG standards in the development of SOA-based IoT systems. *J. Syst. Softw.* **2020**, *169*, 110720. [CrossRef]
94. Costa, B.; Pires, P.F.; Delicato, F.C. Modeling SOA-Based IoT Applications with SoaML4IoT. In Proceedings of the IEEE 5th World Forum on Internet of Things (WF-IoT), Limerick, Ireland, 15–18 April 2019.
95. Gomes, T.; Lopes, P.; Alves, J.; Mestre, P.; Cabral, J.; Monteiro, J.L.; Tavares, A. A modeling domain-specific language for IoT-enabled operating systems. In Proceedings of the 43rd Annual Conference of the IEEE Industrial Electronics Society, Beijing, China, 29 October–1 November 2017.
96. Sivieri, A.; Mottola, L.; Cugola, G. Building Internet of Things software with ELIoT. *Comput. Commun.* **2016**, *89*, 141–153. [CrossRef]
97. Asici, T.Z.; Karaduman, B.; Eslampanah, R.; Challenger, M.; Denil, J.; Vangheluwe, H. Applying Model Driven Engineering Techniques to the Development of Contiki-Based IoT Systems. In Proceedings of the IEEE/ACM 1st International Workshop on Software Engineering Research & Practices for the Internet of Things, Montreal, QC, Canada, 27 May 2019.

98. Sosa-Reyna, C.M.; Tello-Leal, E.; Lara-Alabazares, D. An Approach Based on Model-Driven Development for IoT Applications. In Proceedings of the IEEE International Congress on Internet of Things, San Francisco, CA, USA, 2–7 July 2018.
99. Sosa-Reyna, C.M.; Tello-Leal, E.; Lara-Alabazares, D. Methodology for the model-driven development of service oriented IoT applications. *J. Syst. Archit.* **2018**, *90*, 15–22. [CrossRef]
100. Ferry, N.; Nguyen, P.; Song, H.; Novac, P.E.; Lavirotte, S.; Tigli, J.Y.; Solberg, A. GeneSIS: Continuous Orchestration and Deployment of Smart IoT Systems. In Proceedings of the IEEE 43rd Annual Computer Software and Applications Conference, Milwaukee, WI, USA, 15–19 July 2019.
101. Ferry, N.; Nguyen, P.H. Towards Model-Based Continuous Deployment of Secure IoT Systems. In Proceedings of the ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), Munich, Germany, 15–20 September 2019.
102. Conzon, D.; Rashid, M.R.A.; Tao, X.; Soriano, A.; Nicholson, R.; Ferrera, E. BRAIN-IoT: Model-Based Framework for Dependable Sensing and Actuation in Intelligent Decentralized IoT Systems. In Proceedings of the 4th International Conference on Computing, Communications and Security, Rome, Italy, 10–12 October 2019.
103. Rafique, W.; Zhao, X.; Yu, S.; Yaqoob, I.; Imran, M.; Dou, W. An Application Development Framework for Internet-of-Things Service Orchestration. *IEEE Internet Things J.* **2020**, *7*, 4543–4556. [CrossRef]
104. Iovino, L.; Sanctis, M.D.; Rossi, M.T. Automated Code Generation for NFC-based Access Control. In Proceedings of the 4th International Workshop on Model-Driven Engineering for the Internet-of-Things, Online, 22 June 2020.
105. Patel, P.; Cassou, D. Enabling high-level application development for the Internet of Things. *J. Syst. Softw.* **2015**, *103*, 62–84. [CrossRef]
106. Soukaras, D.; Patel, P.; Song, H.; Chaudhary, S. IoTSuite: A ToolSuite for Prototyping Internet of Things Applications. In Proceedings of the 4th Workshop on on Computing and Networking for Internet of Things (ComNet-IoT 2015), Goa, India, 4–7 January 2015.
107. Cueva-Fernandez, G.; Espada, J.P.; García-Díaz, V.; García, C.G.; Garcia-Fernandez, N. Vitruvius An expert system for vehicle sensor tracking and managing. *J. Netw. Comput. Appl.* **2014**, *42*, 178–188. [CrossRef]
108. García, C.G.; Bustelo, B.C.P.G.; Espada, J.P.; Cueva-Fernandez, G. Midgar: Generation of heterogeneous objects interconnecting applications, A Domain Specific Language proposal for Internet of Things scenarios. *Comput. Netw.* **2014**, *64*, 143–158. [CrossRef]
109. García, C.G.; Espada, J.P.; Núñez-Valdez, E.R.; García-Díaz, V. Midgar: Domain-Specific Language to generate Smart Objects for an Internet of Things platform. In Proceedings of the Eighth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, Birmingham, UK, 2–4 July 2014.
110. Schachinger, D.; Kastner, W. Model-driven integration of building automation systems into Web service gateways. In Proceedings of the IEEE World Conference on Factory Communication Systems (WFCS), Palma de Mallorca, Spain, 27–29 May 2015.
111. Karkouch, A.; Mousannif, H.; Moatassime, H.A.; Noel, T. A model-driven architecture-based data quality management framework for the internet of Things. In Proceedings of the 2nd International Conference on Cloud Computing Technologies and Applications (CloudTech), Marrakech, Morocco, 24–26 May 2016.
112. Pramudianto, F.; Eisenhauer, M.; Kamienski, C.A.; Sadok, D.; Souto, E.J. Connecting the Internet of Things rapidly through a model driven approach. In Proceedings of the IEEE 3rd World Forum on Internet of Things (WF-IoT), Reston, VA, USA, 12–14 December 2016.
113. Pramudianto, F.; Kamienski, C.A.; Souto, E.; Borelli, F.; Gomes, L.L.; Sadok, D.; Jarke, M. IoT Link: An Internet of Things Prototyping Toolkit. In Proceedings of the IEEE 11th Intl Conf on Ubiquitous Intelligence and Computing and IEEE 11th Intl Conf on Autonomic and Trusted Computing and IEEE 14th Intl Conf on Scalable Computing and Communications and Its Associated Workshops, Bali, Indonesia, 9–12 December 2014.
114. Costa, B.; Pires, P.F.; Delicato, F.C. Modeling IoT Applications with SysML4IoT. In Proceedings of the 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Limassol, Cyprus, 31 August–2 September 2016.
115. Hussein, M.; Li, S.; Radermacher, A. Model-driven Development of Adaptive IoT Systems. In Proceedings of the 4th International Workshop on Interplay of Model-Driven Engineering and Component-Based Software Engineering, Saint-Malo, France, 2 October 2016.
116. Nguyen, X.T.; Tran, H.T.; Baraki, H.; Geihs, K. FRASAD: A framework for model-driven IoT Application Development. In Proceedings of the IEEE 2nd World Forum on Internet of Things (WF-IoT), Milan, Italy, 14–16 December 2015.
117. Eclipse Vorto Project. Available online: https://www.eclipse.org/vorto/ (accessed on 27 October 2022).
118. Dautov, R.; Song, H. Towards IoT Diversity via Automated Fleet Management. In Proceedings of the 4th International Workshop on Model-Driven Engineering for the Internet-of-Things, Online, 22 June 2020.
119. ENACT Project. Available online: https://www.enact-project.eu/ (accessed on 27 October 2022).
120. Betancourt, V.P.; Liu, B.; Becker, J. Model-based Development of a Dynamic Container-Based Edge Computing System. In Proceedings of the IEEE International Symposium on Systems Engineering, Vienna, Austria, 12 October–12 November 2020.
121. Tufail, H.; Azam, F.; Anwar, M.W.; Zafar, M.N.; Muzaffar, A.W.; Butt, W.H. A Model-Driven Alarms Framework (MAF) With Mobile Clients Support for Wide-Ranging Industrial Control Systems. *IEEE Access* **2020**, *8*, 174279–174304. [CrossRef]
122. Hassine, T.B.; Khayati, O.; Ghezala, H.B. An IoT domain meta-model and an approach to software development of IoT solutions. In Proceedings of the International Conference on Internet of Things, Embedded Systems and Communications (IINTEC), Tunis, Tunisia, 20–22 December 2019.

123. Neisse, R.; Fovino, I.N.; Baldini, G.; Stavroulaki, V.; Vlacheas, P.; Giaffreda, R. A Model-based Security Toolkit for the Internet of Things. In Proceedings of the 9th International Conference on Availability, Reliability and Security, Washington, DC, USA, 8–12 September 2014.
124. Plotkin, G.D. The origins of structural operational semantics. *J. Log. Algebraic Program.* **2004**, *61*, 3–15. [CrossRef]
125. Schmidt, D.A. *Denotational Semantics: A Methodology for Language Development*; William C. Brown Publishers: Dubuque, IA, USA, 1986.
126. Hervey, S.G.J. *Axiomatic Semantics: A Theory of Linguistic Semantics*; Scottish Academic Press: Edinburgh, UK, 1979.
127. Murata, T. Petri nets: Properties, analysis and applications. *Proc. IEEE* **1989**, *77*, 541–580. [CrossRef]
128. Pnueli, A. The temporal logic of programs. In Proceedings of the SFCS'77 18th Annual Symposium on Foundations of Computer Science, Washington, DC, USA, 31 October–2 November 1977.
129. Kosar, T.; Bohra, S.; Mernik, M. Domain-Specific Languages: A Systematic Mapping Study. *Inf. Softw. Technol.* **2016**, *71*, 77–91. [CrossRef]
130. Semantics of a Foundational Subset for Executable UML Models (fUML), An OMG Executable UML Publication. Available online: https://www.omg.org/spec/FUML/1.5/PDF (accessed on 29 October 2022).
131. Frank, U. Multilevel Modeling. *Bus. Inf. Syst. Eng.* **2014**, *6*, 319–337. [CrossRef]
132. Kos, T.; Mernik, M.; Kosar, T. A tool support for model-driven development: An industrial case study from a measurement domain. *Appl. Sci.* **2019**, *9*, 4553. [CrossRef]
133. Kos, T.; Mernik, M.; Kosar, T. Evolution of Domain-Specific Modeling Language: An Example of an Industrial Case Study on an RT-Sequencer. *Appl. Sci.* **2022**, *12*, 12286. [CrossRef]
134. The nuXmv Model Checker. Available online: https://nuxmv.fbk.eu/ (accessed on 29 October 2022).
135. Eickhoff, C.; Lange, M.; Raesch, S.L.; Zündorf, A. EMFeR: Model Checking for Object Oriented (EMF) Models. In Proceedings of the 7th International Conference on Model-Driven Engineering and Software Development, Setubal, Portugal, 20–22 February 2019.
136. Holzmann, G.J. The model checker SPIN. *IEEE Trans. Softw. Eng.* **1997**, *23*, 279–295. [CrossRef]
137. Magee, J.; Kramer, J. *Concurrency—State Models and Java Programs*, 2nd ed.; Wiley: Hoboken, NJ, USA, 2006.
138. Lethbridge, T.C.; Laganiere, R. *Object-Oriented Software Engineering: Practical Software Development Using UML and Java*, 2nd ed.; McGraw Hill Higher Education: London, UK, 2004.
139. Skelton, M.; Pais, M. *Team Topologies: Organizing Business and Technology Teams for Fast Flow*; IT Revolution Press: Portland, OR, USA, 2019.
140. Wohlin, C.; Runeson, P.; Höst, M.; Ohlsson, M.C.; Regnell, B.; Wesslen, A. *Experimentation in Software Engineering*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2012.
141. Ozkaya, M.; Erata, F. Understanding Practitioners' Challenges on Software Modeling: A Survey. *J. Comput. Lang.* **2020**, *58*, 100963. [CrossRef]