



ISSN: 0067-2904

Developing a Graphical Domain-Specific Modeling Language for Efficient Lightweight Block Cipher Schemas Configuration: LWBCLang

Samar Amil Qassir^{1*}, Methaq Talib Gaata¹, Ahmed T. Sadiq², Imad Fakhri Taha³

¹Department of Computer Science, College of Science, Mustansiriyah University, Baghdad, Iraq

²Department of Computer Science, University of Technology-Iraq, Baghdad, Iraq

³Department of Computer Science, College of Science, Düzce University, Düzce, Turkey

Received: 24/5/2023 Accepted: 1/9/2023 Published: xx

Abstract

The lightweight block cipher is an encryption technique with negligible computational overhead. Despite its advantages, it faces a substantial challenge. Correct handwriting of the script code for the cipher scheme is a challenge for programmers. In this research, we suggest a new graphical domain-specific modeling language to make it easier for both non-technical users and domain specialists to implement lightweight block cipher schemes. The proposed language, called LWBCLang, is a modular and extensible language that offers graphical components for constructing three essential types of inner block cipher structures. Seven different methods of keystream generation and all the tests of the NIST suite with performance analysis are provided. In the context of its meta-model, LWBCLang's abstract and concrete syntaxes are specified. LWBCLang has been implemented as an internal DSML with Python as the host language. The evaluation of LWBCLang is based on qualitative analysis to demonstrate the language's effectiveness and efficiency. Further benefits of this proposed language are evaluated and discussed in depth in this research.

Keywords: Cipher Structure, Domain Specific Modeling Language, Symmetric cipher, Lightweight Block Cipher, Meta-Model.

تطوير لغة نمذجة رسومية خاصة بتكوين نماذج لخوارزميات تشفير الكتلة خفيفة الوزن:

LWBCLang

سمر اميل يوسف^{1*}, ميثاق طالب¹, احمد طارق², عماد فخري طه³

¹ قسم علوم الحاسوب، كلية العلوم، الجامعة المستنصرية، بغداد، العراق

² قسم علوم الحاسوب، كلية العلوم، الجامعة التكنولوجية، بغداد، العراق

³ قسم علوم الحاسوب، كلية العلوم، جامعة دوزجي، دوزجي، تركيا

الخلاصة

تشفير الكتلة خفيف الوزن هو تقنية تشفير ذات عبء حسابي ضئيل. على الرغم من مزاياه العديدة، إلا أنه يواجه تحديًا كبيرًا. تمثل الكتابة اليدوية الصحيحة للكود النصي لنظام التشفير تحديًا للمبرمجين.

*Email: samarqassir@uomustansiriyah.edu.iq

في هذا البحث ، نقتراح لغة نمذجة رسومية جديدة خاصة بمجال التشفير لتسهيل الأمر على المستخدمين غير التقنيين والمتخصصين في المجال لتنفيذ مخططات تشفير الكتلة خفيفة الوزن. اللغة المقترحة ، المسماة LWBCLang ، هي لغة معيارية وقابلة للتوسيع توفر مكونات رسومية لبناء الأنواع الثلاثة الأساسية من هياكل تشفير الكتلة الداخلية. يتم توفير سبع طرق مختلفة لتوليد تدفق المفاتيح وجميع اختبارات مجموعة NIST مع تحليل الأداء. في سياق meta-model الخاص به ، تم تحديد بناء جمل LWBCLang المجرد والملموس. تم تطوير LWBCLang باعتباره DSML داخليًا مع Python كلفة مضافة. يعتمد تقييم LWBCLang على التحليل النوعي لإثبات فعالية اللغة وكفاءتها. يتم تقييم الفوائد الإضافية لهذه اللغة المقترحة ومناقشتها بعمق في هذا البحث.

1. Introduction

An encryption technique called the lightweight cipher (LWC) offers anonymity for particular purposes. This cipher type was created for applications with a high rate of growth and heavy reliance on low-resource hardware like smartcards, the internet of things (IoT), the wireless body area network (WBAN), and wireless sensor networks (WSN) [1, 2]. Applications often share private or sensitive data; therefore, ensuring a sufficient level of data security is a vital necessity. According to the historical order shown in Figure 1, this cipher method is categorized as belonging to one group of cipher types [3].

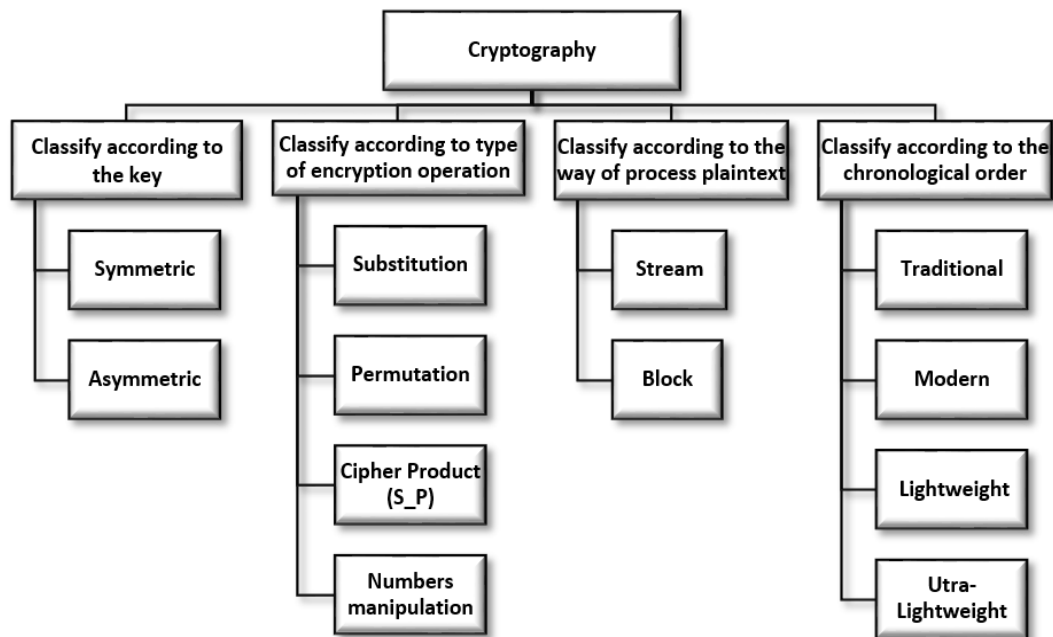


Figure 1: The Cipher Taxonomy [3]

The classification of ciphers may use more than one of the categorization methods stated above; for instance, the cipher may be symmetric, block-processing, or lightweight block cipher (LWBC) [3]. There are three fundamental categories of LWBC based on their internal structures: substitution-permutation networks (SPN), general-feistel networks (GFN), and add-rotate-XOR (ARX). The SPN is a type of product cipher that, at each round, combines the permutation layer with the substitution layer for the diffusion process. In each round of the GFN cipher structure, the input state is split into two equal parts (the u_i and v_i branches). The round function is applied to one half of the input state, which is then processed with the other half of the data (the target) using logic gates before the two parts are switched. ARXs use modular addition, rotation, and logic operation XOR without using S-boxes. They produce compact and fast implementations for software and hardware implementations [4–6]. In this research, we

chose the KLEN [7] family, which is based on SPN structure, the GOST [8] family, which is based on GFN [9], and the SPECK [10] family, which is based on ARX structure, as case studies for the LWBC domain to implement in the proposed language. The LWBC can be implemented using either hardware or software. Writing practical and effective cipher programs presents obstacles (software issues) to software implementation utilizing GPPLs.

In our previous work [11], we defined a graphical DSML "SCLang" that significantly increases the flexibility, expressiveness, and ease of stream cipher schema design and implementation. The first version of our DSML was for both beginner and expert programmers. It shows in a diagram how to put together the main parts of the stream cipher schema: six different ways to make a keystream that can be used in a hybrid way (one or multiple levels), four logic gates, and fifteen tests from the NIST suite that make it easier to do a statistical analysis of encrypted results. The abstract syntax of SCLang consists of five packages, along with its restrictions based on domain concepts. For the concrete syntax, meaningful icons for meta-elements were chosen in addition to the static type used to define the semantics. The proposed SCLang makes it easier to generate a random sequence and test it by providing a higher level of abstraction, generating the random sequence automatically, improving the performance of the cipher schema (in both design and implementation), and making things run more smoothly by making mistakes less likely.

The contribution of this research is to design and implement LWBCLang, a graphical DSML for the LWBC domain. It hides the details of coding and provides a high level of abstraction when configuring the cipher schema. By providing the main components of the LWBC domain with seven different keystream generation methods that can be used for seed random key generation, the language provides the programmer with the ability to construct cipher schemas in a flexible manner. In addition, provide the statistical and performance tests for results analysis. In short, LWBCLang provides flexible and automatic transformation of plaintext into the corresponding ciphertext using graphical components.

The remaining portions of this research are structured as follows: Highlighted in Section 2 are the DSLs and DSMLs. In Section 3, some relevant DSML work is introduced. The architecture of the proposed LWBCLang is described in Section 4. Section 5 discusses implementation specifics and presents some implemented examples. The evaluation of the proposed language is described in Section 6, and finally, Section 7 offers concluding observations and enumerates the key components of the language that has been delivered.

2. Domain Specific Language

DSLs are software languages that offer the abstractions needed to describe a system or model whose expressiveness is restricted to a narrowly specified area. When compared to GPPLs in their field of application, they provide significant improvements in expressiveness and usability [12, 13]. As a result, DSLs have grown in popularity as a new area of study in the field of software engineering (SE) and as a key component of several software development methodologies, including generative programming (GP), product lines (PL), software factories (SF), and model-driven engineering (MDE). More methods than GPPLs may be used to develop DSLs; they could develop as external or internal languages [14, 15]. The mapping process between the abstract and concrete syntax of the DSLs can be either textual or graphical. As with the DSEL for embedded language, the DSML for modeling, and the DSVL for visualization, the specific functionalities that the DSLs highlight are also primarily related to the intended domains for which they were developed. Similar to traditional GPPLs, DSLs are expressed using the three implementation concerns of abstract syntax, concrete syntax, and semantics, as

explained in Figure 2. The set of language concepts relevant to a DSL's domain, as well as the relationships between them, are defined by the language's abstract syntax [15, 16]. This abstract syntax is translated into a collection of (textual or graphic) symbols by its concrete syntax, which programmers use to build programs and models. The language editor enables programmers to write programs or configure models (schemas) via the graphical notations outlined by the concrete syntax. Additionally, each of a DSL's linguistic constructions has a distinct meaning according to the language's semantics. In more specific terms, static semantics limits the categories of valid programs and models, whereas dynamic semantics provides the criteria by which they are assessed during execution [17]. When compared to the conventional software development process, which employs GPLs such as Java or C++, empirical research has demonstrated that productivity rises with DSL usage [15].

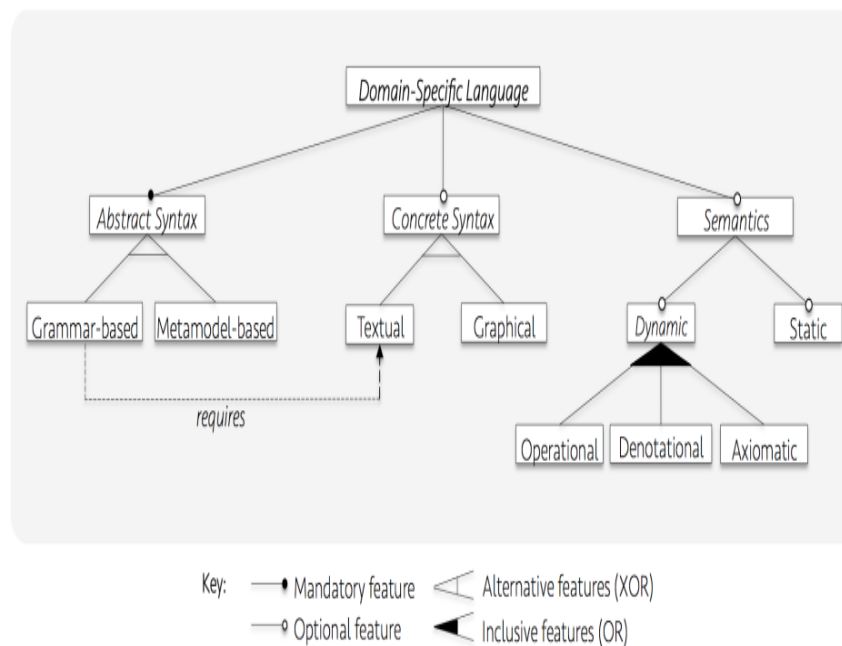


Figure 2: DSL definition [16]

A programmer who uses a general-purpose programming language (GPL) is able to create a program in any field for a wide range of application domains. But each GPL has its difficulties; some of them are sensitive to space, small, or capital letters [16]. A program's design is a genuine difficulty; even a small program can require the naming of many things like variables, procedures, functions, classes, objects, etc. Thus, if the programmer is a beginner, he needs to first learn the syntax of that GPL before trying to write the code, debug all bugs, and implement the program. In comparison to GPLs, DSLs provide various benefits for expressing a particular domain. One benefit is that it offers greater abstractions for the target domain, increasing output and improving the standard of the development process. The presented LWBCLang language alleviates the programming complexity of the GPLs through the use of simplified interfaces; it provides interactive visualizations; and the user-friendly, common interactive GUI with drag-and-drop capability allows for fruitful and interactive use for creating and implementing a wide range of domain schemas.

3. Related Work

Some studies that have developed and used DSML as a remedy for specific problems in application domains are discussed [17–19]. To express language learning processes, Sebastián et al. [20] provided a graphical notation for a domain-specific language. It explains how this

notation lets programmers talk about workflow, presentation, content, media, and activity models by following a meta-model that specifies the abstract syntax of the domain-specific language to reflect how language learning activities work. In order to create model-based applications, this notation is implemented as a component of an integrated development environment. The approach used to analyze this proposal employs the cognitive aspects of notational systems. The reflexive model editor is inferior to the suggested visual diagram editor in terms of user experience. In comparison to the conventional reflexive tree notation used by many model-based development frameworks, the suggested graphical notation is more intuitive and simple to maintain visually for the generation and maintenance of workflow models and presentation/activity models.

Sadik and Geylani [21] proposed a new DSML called DSML4DT for device trees (DTs). DTs include node specifications as well as descriptions of the devices and peripherals located inside an embedded system. The development of DT models using the DSML4DT language enabled designers to visually design and build embedded systems based on DT. DSML4DT's meta-model was made up of more than 70 meta-entities and their connections. Based on DSML4DT model validation rules built on the Sirius platform with the help of Acceleo Query Language (AQL), the environment was given automatic constraint checks and static semantic restrictions.

Model-driven (MD) concepts and DSMLs were used in research by Vjetica et al. [22] to introduce a framework for the formal description and automated execution of manufacturing processes. In this manner, manufacturing process models serve as the primary management tools. In this study, the production process modeling space was examined, and a DSML was developed that may be used to build production process models appropriate for automatic code generation. The resulting code is used to automate manufacturing operations on the shop floor or in a simulation. The language may be used to indicate potential faults that could arise while the process is running, as well as error handling and remedial actions. The DSML was assessed by several user groups.

Graphical Invasive Language (GIRL) was introduced by Marzina et al. [23]. GIRL is a DSL based on set theory used to express the structural invariants of software requirements. The purpose of the proposed language's design is to provide a straightforward visual language based on set notation where demand limitations can be automatically analyzed. The Meta-Object Facility (MOF) meta_model, consisting of items such as an integer, an operation, and their connections, provides the GIRL abstract syntax. The alloy analyzer can assess the consistency of the structural requirement using translational semantics without user input. This translational technique offers the advantages of formal analysis by enabling the early discovery of discrepancies in the requirement definitions. Ten volunteer software engineers participated in empirical research to evaluate GIRL and its automated analysis. Participants did not report any issues utilizing entities or relationships as a consequence. However, the automated analysis enabled them to spot errors, and nine out of ten, they accurately defined all requirements.

Ana et al. [24] suggested a graphical DSML to help domain specialists retrieve event logs from ERP systems. In particular, domain experts are able to conceptually locate where instances and events are stored inside a database. Following automated validation, these conceptual models are converted into SQL code. This modeling language was designed to address complicated conditions when using ERP systems. The modeling language's applicability was demonstrated via a case study with actual data to show that the language contained the necessary components. These constructs make it possible for domain specialists to focus on

modeling data during the log extraction stage without learning how to program, which simplifies the querying of process data.

4. The proposed LWBCLang

The LWBCLang is thoroughly defined in this section and its subsections. The first step is to define abstract syntax using a meta-model that is based on LWBC domain concepts and the rules that control them during the encryption and decryption procedures. Second, expressive icons are chosen to represent the notions in the LWBC domain, which defines the concrete syntax. In order to warn the programmer and handle any missing or incorrect connections that may have occurred during the building of the cipher schema, the semantics are finally defined. All of these definitions work together to create a valid LWBCLang's meta_model, which is then implemented as an internal graphical DSML that takes advantage of Python's strengths (the host language) and functions as a graphical editor. Figure 3 explains the definition steps of this proposed LWBCLang language.

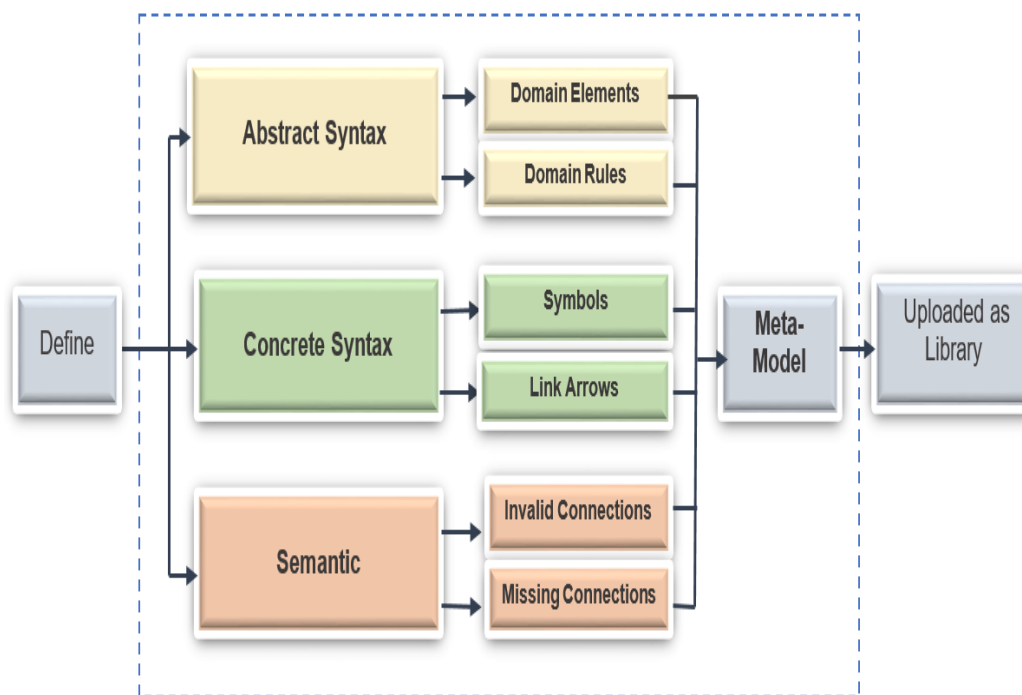


Figure 3: Definition steps of the LWBCLang language

In the proposed LWBCLang language, we provide the following contributions based on utilizing DSML:

- 1- A new graphical DSML for the LWBC domain is proposed that significantly increases the flexibility, expressiveness, and ease of constructing LWBC schemas.
- 2- The proposed LWBCLang has graphical building blocks for three families in the LWBC domain: the KLEIN, GOST, and SPECK families for the SPN, GFN, and ARX types of inner cipher structures.
- 3- The proposed LWBCLang has seven different keystream generation methods (Geffe, Random Shuffled, Linear Feedback Shift Register (LFSR), Non-Linear Feedback Shift Register (NLFSRF), Non-Linear Feedback Shift Register (NLFSRG), Salsa20_based, and A5/1_based) that can be used to create seed keys for LWBC schemas. More information about these methods can be found in [25].
- 4- The proposed LWBCLang includes all of the tests in the NIST suite. The details of these tests are shown in [25] as graphical parts to make statistical analysis of encrypted results easier.

5- The proposed LWBCLang provides performance analysis that computes encryption time, decryption time, number of encrypted blocks, entropy, and throughput.

The proposed language provides the programmer with a user-friendly interactive interface that consists of two sides: the left is the component side, and the right is the workspace side. The component side consists of seven sections. Each section represents one package in the proposed meta-model and contains a number of meta-elements (components) that represent the concepts of the LWBC domain. These seven sections are explained as follows:

- 1- CommonComponents consists of nine components (Plaintext, Ciphertext, Split, Combined, left part, right part, ToBlock, key size, and Number of rounds).
- 2- GFN consists of three components (Rounds, Key generation, and Decryption).
- 3- SPN consists of four components (ToState, Rounds, Key generation, and Decryption).
- 4- ARX consists of three components (Rounds, Key generation, and Decryption).
- 5- Performance consists of two components (Analyzer and a Performance tester).
- 6- SeedKeyMethods has seven parts: Geffe, linear feedback shift register (LFSR), random shuffled, A5/1_based, non-linear feedback shift register Fibonacci (NLFSRF), non-linear feedback shift register Galois (NLFSRG), and Salsa20_based.
- 7- Test consists of all NIST tests.

i. Abstract syntax of LWBCLang

The identification of the domain concepts and their relations provides the basis for constructing a DSML. These concepts, their relations, and the constraints that go along with them are proposed using a meta-model. To express a language's abstract syntax, a meta-model is created using the Unified Modeling Language (UML) package diagram. The LWBCLang meta-model is built using the notions of the LWBC domain. Figure 4 shows that the meta-model is split into eight packages. Each package has a number of related meta-elements that represent all LWBC domain concepts and their connections for the three families (KLEIN, GOST, and SPECK) and the three different types of inner cipher structures (SPN, GFN, and ARX). The first package, GraphicalEnvironment, of the proposed language defines every element of the implementation of the graphical environment. These graphics are built using outside libraries (PyQt5, Matplotlib, and Orange Canavas). The next seven packages' specifics are provided in Tables (1-4).

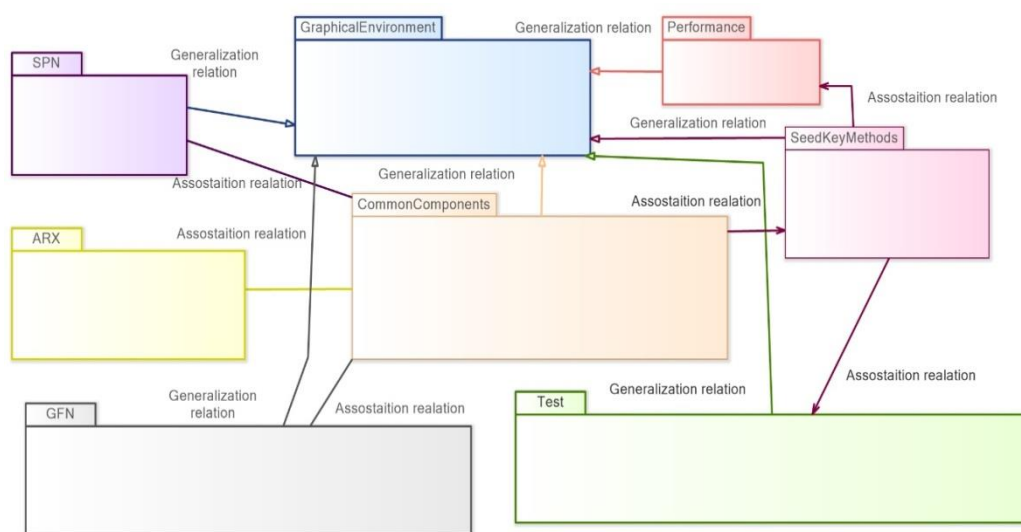


Figure 4: Meta-model of LWBCLang

Table 1: Description of meta-elements in the CommonComponents package

Meta-element	Side link		Number of links	Accept	Produce
Plaintext	Left	-	-	-	bitList bitLength
	Right	Can be linked to ToBlock meta-element	Many		
	Details	It is the main meta-element, the root of every LWBC schema will be start by this component. It accepts plain text by manually or through load file of these types (*.doc, *.pdf, *.txt).			
Ciphertext	Left	Can be linked to Rounds meta-element in ARX, GFN and SPN packages	One	bitList	bitList bitLength
	Right	Can be linked to any meta-element in NIST tests package	Many		
	Details	It is the meta-element used for every LWBC schema as final component, it used to display and save the result of LWBC schema, as one of these file types (*.doc, *.pdf, *.txt).			
ToBlock	Left	Can be linked to the plaintext class in the CommonComponents package.	One	bitList	blocks
	Right	Can be linked to the ToState in the SPN package, Split class in the GFN package, and Into4Parts in the ARX package.	One		
	Details	It is the second class (constituent) that is used for the LWBC schemas; it is used by all GFN, SPN, and ARX packages.			
Split	Left	Can be linked to ToBlock	One	bitList	Two bitLists
	Right	Can be linked to LeftPart, RightPart, Rounds in ARX and GFN	Two		
	Details	This meta-element is used by ARX and GFN packages.			
Combined	Left	Can be linked to LeftPart, RightPart	Two	bitList	Two bitLists
	Right	Can be linked to Ciphertext	One		
	Details	This meta-element is used by ARX and GFN packages.			
LeftPart, RightPart	Left	Can be linked to split meta-element	One	bitList	bitList
	Right	Can be linked to Rounds in ARX and GFN	One		
	Details	This meta-element is used by ARX and GFN packages.			
KeySize	Left	-	-	-	no_k
	Right	Can be linked to any meta-element in Seed Key Method meta-element	One		
	Details	This meta-element is used by GFN, SPN, ARX packages, it used to determine the key size of LWBC schema.			
NumberRounds	Left	-	-	-	no_r
	Right	Can be linked to Rounds, KeyGen, and Decrypted meta-elements IN GFN	Three		
	Details	This meta-element is used by GFN package, it used to determine the rounds number of LWBC schema.			

Table 2: Description of meta-elements in the GFN package

Meta-element	Side link		Number of links	Accept	Produce
Rounds	Left	Can be linked to Left Part, RightPart, NumberRounds, meta-elements in CommonComponents package and KeyGen meta-element in GFN package.	Many	bitLists (Left and Right), no_r, subkey_r	bitLists (Left and Right), Per_Infor
	Right	Can be linked to LeftPart and RightPart meta-elements in CommonComponents package	Two		
	Details	This meta-element is used by GFN LWBC schema. It is performing the encryption process.			
Decrypted	Left	Can be linked to KeyGen, Rounds meta-elements in GFN and NumberRounds meta-element in CommonComponents package, Performance meta-element in Performance package.	Many	bitList, no_r, subkey_r	bitList, Per_Infor
	Right	Can be linked to performance meta-element in performance package	Two		
	Details	This meta-element is used by GFN FWBC schema. It is performing the decryption process.			
KeyGen	Left	Can be linked to any meta-element in SeedKeyMethods package	One	bitList, no_r	subkey_r
	Right	Can be linked to Rounds, Decrypted in GFN package, NumberRounds in CommonComponents package	Three		
	Details	This meta-element is used by GFN LWBC schemas. It is generated the subkeys for encryption/ decryption processes.			

Table 3: Description of Meta-Elements in the SPN Package

Meta-element	Side link		Number of links	Accept	Produce
ToState	Left	Can be linked to ToBlock meta-element in in CommonComponents package	One	bitList	stateList
	Right	Can be linked to Rounds meta-element	One		
	Details	It is the first meta-element used for SPN LWBC schema to transfer bitList into matrix form.			
Rounds	Left	Can be linked to ToState and KeyGene meta-elements	Two	stateList, subkey_r	CiphstateList, Per_Infor
	Right	Can be linked to Decrypted meta-element and Ciphertext meta-element in CommonComponents package, Performance meta-element in Performance package	Two		
	Details	This meta-element is used by SPN LWBC schema. It is performing the encryption process.			
Decrypted	Left	Can be linked Rounds, KeyGene meta-elements and Performance meta-element in Performance package	Three	Ciphstate List, subkey_r	stateList, Per_Infor

	Right	Can be linked to performance meta-element in performance package	Two		
	Details	This meta-element is used by SPN LWBC schema. It is performing the decryption process.			
KeyGene	Left	Can be linked to any meta-element in SeedKeyMethods package	One	bitList, no_r	subkey_r
	Right	Can be linked to Rounds and Decrypted meta-elements in SPN package	Two		
	Details	This meta-element is used by SPN LWBC schema. It is generated the subkeys for encryption/ decryption processes.			

Table 4: Description of Meta-Elements in the ARX Package

Meta-element		Side link	Number of links	Accept	Produce
Rounds	Left	Can be linked to Left Part, RightPart meta-elements in CommonComponents package and KeyGen meta-element in ARX package.	Many	Two bitLists, subkey_r	CiphbitList, Per_Infor
	Right	Can be linked to ARX_Decrypted meta-element and Ciphertext meta-element in CommonComponents package, Performance meta-element in Performance package.	Two		
	Details	This meta-element is used by ARX LWBC schema. It is performing the encryption process.			
Decrypted	Left	Can be linked to Rounds and KeyGene meta-elements	Two	CiphbitList, subkey_r,	bitLists, Per_Infor
	Right	Can be linked to performance meta-element.	Two		
	Details	This meta-element is used by ARX LWBC schema. It is performing the decryption process.			
KeyGene	Left	Can be linked to any meta-element in SeedKeyMethods package.	One	bitList, no_r	subkey_r
	Right	Can be linked to Rounds and Decrypted meta-elements in ARX package.	Two		
	Details	This meta-element is used by ARX LWBC schema. It is generated the subkeys for encryption/ decryption processes.			

The test package in the meta-model of LWBCLang consists of fifteen meta-elements that represent NIST tests implemented according to their details in [25]. This package is used through directed association relations by the ciphertext meta-element in the CommonComponents package; each meta-element of this package accepts bitList from the ciphertext meta-element through its left-side link, then performs its computation and displays the randomness analysis result.
















The SeedKeyMethods package in the meta-model of LWBCLang consists of seven meta-elements, according to their details in [25]. This package is used by the KeySize meta-element in the CommonComponents package and the KeyGene meta-element in the SPN, GFN, and ARX packages through directed association relations. It gets (bitList and bitLen) from the KeySize meta-element through its left-side link, does its computation, and sends the randomness sequence result to KeyGene through its right-side link.

The last package in the meta-model of LWBCLang is Performance, which consists of two meta-elements (Analyzer and Performance Tester). The analyzer meta-element used through (directed association relation) by any meta-element in the SeedKeyMethods package accepts (bitList and bitLen) from the seed key through its left-side link, then performs its computation (periodicity, balance property, run length property, and autocorrelation) and displays the analysis result. For the performance tester, it is accepting (bitList) from the decrypted meta-element through its left-side link, then performing its computation (encryption time, decryption time, number of encrypted blocks, entropy, and throughput), and displaying the analysis results.

ii. Concrete Syntax of LWBCLang

The definition of “abstract syntax” includes both the ideas that the language represents and the connections between those ideas. Unlike the description of concrete syntax, it gives a mapping between meta-elements and their (graphical or textual) representations. This section covers concrete syntax, which is considered the second design goal, and provides a graphical representation of the proposed LWBCLang. We chose a graphical notation since it can more fully and easily depict a variety of relationships. Table 5 displays these graphical icons that are used for concrete syntax in the language utilized.

Table 5: Some graphical icons of LWBCLang

Concept	Component	Concept	Component	Concept	Component
Plaintext		Performance		LeftPart	
Ciphertext		Test		RightPart	
Decrypted		Analyzer		keystream	
ToBlock		Rounds		ToState	
Combined		Split		KeyGen	

iii. Semantics of LWBCLang

For the provided meta-model and its accompanying meta-elements and links, static semantics are used in the proposed LWBCLang. This method is known as a “restriction check,” and it is used on models (schemas) that may be defined by LWBCLang to stop users from constructing the LWBC schema in the workspace incorrectly. When two components are not connected correctly or are missing from a schema during building, the language reaction is either to display an error message or to reset the workspace to the previous step. The information is provided for each one of these restriction checks as follows:

- 1- Restriction for meta-element numbers. This restriction was put in place to limit the number of components used to construct the LWBC schema. Programmers who attempt to utilize several plaintext components in the same LWBC schema will receive an error notice stating that each LWBC schema should contain a single plaintext.
- 2- Restriction for meta-element relations: for the described meta-model, further types of restrictions are offered for the relationships between the elements. Each connection between

two components in the LWBC schema has a name, and if a link is neglected to be linked between two components, a notification stating the link is missing will be displayed. This constraint regulates the relationship between components.

3- Restriction on the number of meta-element relationships. One more control is applied to the number of relationships among the components in the meta-model based on one-to-one, many-to-one, and one-to-many relationships. One ciphertext component is utilized for each LWBC schema, although the same plaintext component can be used for many LWBC schemas.

4- Restriction on source and destination: This restriction, which controls the relationship's path, identifies the relationship's origin and final destination. Control the relationship's direction to specify the LWBC schema's beginning and conclusion. Before creating the relationship with the plaintext component, the relationship between the (ToBlock) and (Rounds) components cannot be formed. The LWBCLang response to an incorrect attempt is to refresh the workspace and go back one step.

5- Restriction for association-direction relations: This restriction controls the association-direction relationship defined in LWBCLang. Naturally, a meta-element in a meta-model uses another meta-element in one direction. The (ToBlock) component uses the (plaintext) component in one direction. For failed attempts, the LWBCLang response is to refresh the workspace one step back.

5. LWBC Schemas Implementation Examples

The parts that came before this one covered at great length the language's syntax and semantics; this section explains the implementation details and shows examples that are constructed by the proposed language.

With Python as the host language and PyCharm acting as the integrated development environment (IDE), the LWBCLang is created as an internal graphical DSML. The graphical user interfaces were created using the libraries PyQt5, Matplotlib, and Orange Canvas GUI templates. Using the Software Ideas Modeler tool, the LWBCLang meta-model was created. Through the implemented examples, the LWBCLang's viability and utility in a practical environment are demonstrated. We chose these three families since they are among the most well-known LWBC domain algorithms: KLEIN, GOST, and SPECK. The encryption and decryption schemes for the KLEIN LWBC, which is based on the SPN structure, and the GOST cipher scheme, which is based on the GFN structure, are shown in Figures 5 and 6.

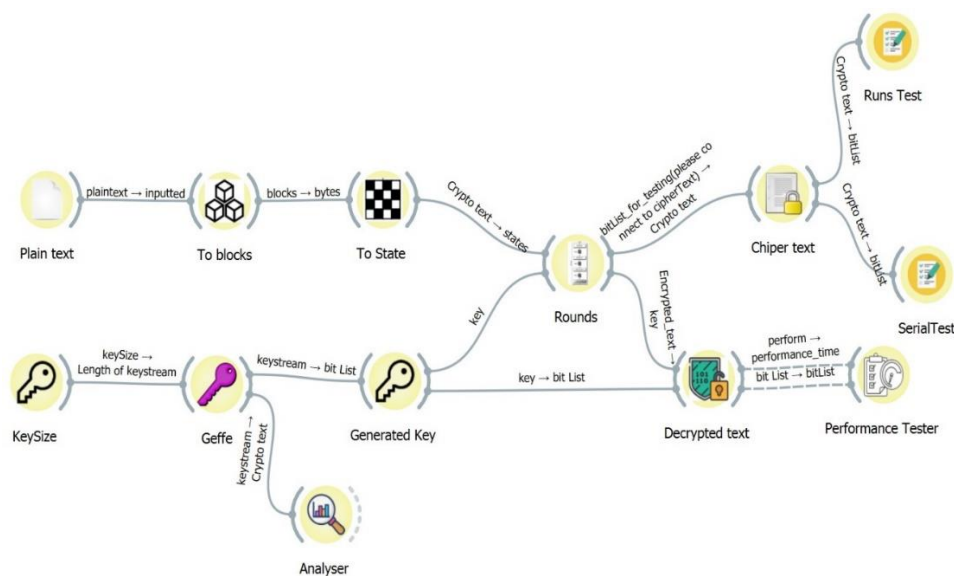


Figure 5: KLEIN LWBC encryption and decryption schema

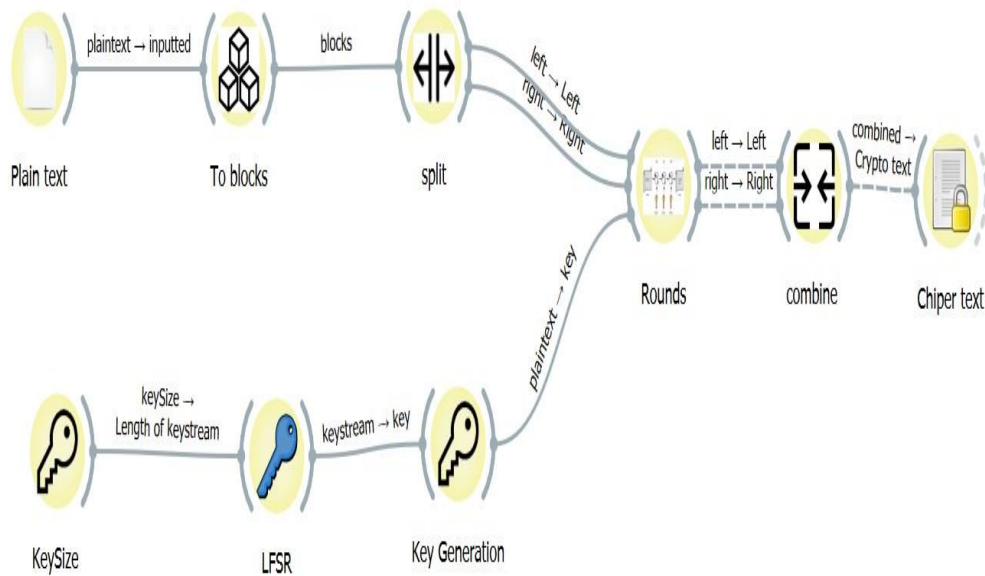


Figure 6: The GOST LWBC encryption schema

6. Evaluation

The assessment employed in this work attempts to evaluate and establish the extent of the proposed language in accordance with the quality and performance standards of graphical modeling languages. Five subjective criteria are employed for a qualitative study of the definition in accordance with metrics [26], and these five criteria are graphical nature, capabilities, ease of understanding, paradigm aid, and extensibility. For each of the aforementioned indicators, further detailed metrics were required. In order to demonstrate what the proposed language achieves, we presented the metrics tables from [26] and highlighted them in light orange as follows:

i. Graphical Nature

The primary components of the ideal graphical language must all be naturally visible. The term "visual" in this sense refers to charts, diagrams, and icons, as well as the skillful use of color and spatial organization. As shown in Table 6, five distinct subjective measures that are more practical can be inferred from the qualitative assessment. The outcomes of our proposed language were as follows: the initial metric was mostly visual; diagrams with meaningful symbols for the second metric; over the entire language for the third; useful use throughout for the fourth; and useful use throughout for the last metric.

Table 6: Measures to evaluate the graphical nature [26]

Assess	highest score					Lowest score
Graphics use	completely graphic, such as iconic	mostly visual	only a little graphic with text commentary	Text with visual embellishments	completely textual	
Graphic used type	Diagrams with meaningful symbols	Icons and a diagram that are largely meaningful	less important symbols and diagrams	Simple forms or graphics	No images	
Graphic use Thoroughness	over the entire language	Generally applicable to most semantics	Approximately half of meanings are applicable	Applied to only a few meanings	No images	
Utilizing space effectively	Useful use throughout	Useful in several aspects	Average efficiency	a minimally effective spatial range	Arrangement of space is insufficient	
Effectiveness of color use	Useful use throughout	Using color effectively in some situations	using color to make a distinction	a minimal use of color	No use of color or improper use of color	

i. Capabilities

The term “capabilities” refers to the language's broad applicability rather than its confinement to a single sector of use. On the basis of the qualitative assessment, as explained in Table 7, two metrics may be established. The outcomes of the proposed language were as follows: the first metric (specific intent) and the second metric (a few domains).

Table 7: Measures to evaluate the capabilities [26]

Assess	highest score					Lowest score
Functioning perfection	a general purpose capability	lacking some capabilities	Several but not all places are affected	Applicable to several areas	specific intent	
Naturalness of implementation	to all domains	to most domains	to many domains	to several domains	a few domains	

ii. Ease of Understood

This measure refers to how easily programs written in the language may be understood. The premise behind visual programming languages is that their graphical nature should make programs easier to understand. If the language doesn't satisfy this need, all we'll be forced to do is learn a new notation for writing programs, which won't solve the software issue. A graphical DSML that is intended for a specific audience is more likely to be successful in boosting understanding than text-based languages. The qualitative evaluation depends on five metrics, which are described in Table 8. The outcomes of our proposed language were much easier than comparison language for the first and much easier for the rest of the metrics.

Table 8: Measures to evaluate the ease of understanding [26]

Assess	highest score				Lowest score
Ease of understood	Much easier than comparison language	Moderately	Moderately	Moderately	Much less than comparison language
Ease for programmers	Much easier	Moderately	Moderately	Moderately	Much less
Simplicity for non-technical programmers	Much easier	Moderately	Moderately	Moderately	Much less
Expert user	Much easier	Moderately	Moderately	Moderately	Much less

iii. Paradigm aid

This measure indicates how well a language supports the programming paradigm for which it is intended. Two metrics can be derived from the qualitative judgment, as explained in Table 9. The outcomes of our proposed language were (very limited) for the first metric and (support for one paradigm) for the second metric.

Table 9: Measures to evaluate the paradigm aid [26]

Assess	highest score				Lowest score
Support for a paradigm	Strong	Moderate	Some support	Weak	very limited
domain support of	all paradigms	several paradigms	some paradigms	few paradigms	one paradigm

iv. Extensibility

This metric refers to the capacity of the language to write large and complex programs. The inextensibility of modern software development processes, methodologies, and tools has been one of the main problems in software engineering for the past 25 years. Four metrics can be derived from the qualitative judgment, as explained in Table 10. The outcomes of our proposed language were strong for all metrics.

Table 10: Measures to evaluate extensibility [26]

Assess	highest score				Lowest score
support modularity for	Strong	Moderate	Some aid	Weak	Nothing at all
support abstraction for	Strong	Moderate	Some aid	Weak	Nothing at all
Support information concealing for	Strong	Moderate	Some aid	Weak	Nothing at all
support for data encapsulation	Strong	Moderate	Some aid	Weak	Nothing at all

The presented LWBCLang is a new DSML in a graphical manner for the LWBC domain. Table 11 presents a short comparison with the traditional GPPL that is used for software implementation in the LWBC domain.

Table 11: Comparison between traditional and the DSML programming language

Programming approach	Abstract syntax	Concrete syntax	No. of programs	Purpose	Paradigm	Execution method
Traditional approach, the GPPL like C#, Python, Java	Grammar-based	texture	One program in each file	General	Procedure/OOP	Compiled/Interpreted
Modern DSML Approach	Grammar/Model	Texture/Graphical	One/Many Schema in each file	Specific	OOP	External/Internal

7. Conclusions

In this research, a new graphical DSML called LWBCLang has been developed for the LWBC cipher domain and is intended for both non-technical users and domain experts. To this purpose, a meta-model has been supplied as the domain's abstract syntax with reference to LWBC concepts and their relations among one another. Also, the concrete syntax of the language has been made available as graphical, meaningful icons. This lets programmers of LWBC schemas make models in LWBCLang for the three basic types of inner cipher structures: SPN, GFN, and ARX, as well as for the three families: KLEIN, GOST, and SPECK. It must be noted that static semantic controls for LWBCLang, within the framework of a number of restrictions, are taken into consideration for explaining the meaning of this new language. For seed-key generation, seven distinct techniques were offered. Last but not least, three kinds of cipher analysis were offered: a set of performance analyses, an analyzer for seed key generation, and comprehensive NIST tests for statistical analysis. In addition to concealing implementation details, flexibility, and a highly expressive graphical user interface with drag-and-drop functionality, an evaluation based on implemented examples reveals a high level of performance. Users have a straightforward and user-friendly method for creating and implementing LWBC domain schemas using the proposed language.

8. Acknowledgements

The work offered in this paper was supported by Mustansiriyah University (www.uomustansiriyah.edu.iq), which is appreciatively acknowledged.

References

- [1] A. S. Jamil, R. A. Azeez, A. Al-Adhami, and N. F. Hassan, "Multibiometric System with Runs Bits Permutation for Creating Cryptographic key Generation Technique," *Iraqi Journal of Science*, vol. 64, no. 1, pp. 452–468, Jan. 2023. Doi: [10.24996/ijis.2023.64.1.40](https://doi.org/10.24996/ijis.2023.64.1.40).
- [2] M. S. Fadhil, A. K. Farhan, and M. N. Fadhil, "A lightweight AES Algorithm Implementation for Secure IoT Environment," *Iraqi Journal of Science*, vol. 62, no. 8, pp. 2759–2770, Aug. 2021. Doi: [10.24996/ijis.2021.62.8.29](https://doi.org/10.24996/ijis.2021.62.8.29).
- [3] Qassir, Samar & Gaata, Methaq & Sadiq, Ahmed, "Modern and Lightweight Component-based Symmetric Cipher Algorithms: A Review," *Aro-The Scientific Journal of Koya University*, vol. 10, pp. 152-168, 2022. doi:0.14500/aro.11007.
- [4] W. Chen, L. Li, Y. Guo, and Y. Huang, "SAND-2: An optimized implementation of lightweight block cipher," *Integration*, vol. 91, pp. 23–34, Jul. 2023, Doi: [10.1016/j.vlsi.2023.02.013](https://doi.org/10.1016/j.vlsi.2023.02.013)

- [5] A. D. Dwivedi and G. Srivastava, "Security analysis of lightweight IoT encryption algorithms: SIMON and SIMECK," *Internet of Things*, vol. 21, p. 100677, Jan. 2023, Doi: 10.1016/j.iot.2022.100677.
- [6] M. M. Hoobi, "Strong Triple Data Encryption Standard Algorithm using Nth Degree Truncated Polynomial Ring Unit," *Iraqi Journal of Science*, vol. 58, no. 3C, pp. 1760–1771, Nov. 2021.
- [7] P. Singh, B. Agrawal, Rahul Kumar Chaurasiya, and B. Acharya, "Low-area and high-speed hardware architectures of KLEIN lightweight block cipher for image encryption," *Journal of Electronic Imaging*, vol. 32, no. 01, Jan. 2023, Doi: 10.1117/1.jei.32.1.013012.
- [8] H. Najm, H. K. Hoomod, and R. Hassan, "A New WoT Cryptography Algorithm Based on GOST and Novel 5d Chaotic System," *Int. J. Interact. Mob. Technol.*, vol. 15, no. 02, pp. 184–199, Jan. 2021.
- [9] W. Yihan and L. Yongzhen, "Improved Design of DES Algorithm Based on Symmetric Encryption Algorithm," in *2021 IEEE International Conference on Power Electronics, Computer Applications (ICPECA)*, Shenyang, China, 2021, pp. 220–223, Doi: 10.1109/ICPECA51329.2021.9362619.
- [10] G. Avoine and J. Hernandez-Castro, Eds., "Security of Ubiquitous Computing Systems," *Springer Nature*, 2021. Doi: 10.1007/978-3-030-10591-4.
- [11] Samar Amil Qassir, Methaq Talib Gaata, and A. T. Sadiq, "SCLang: Graphical Domain-Specific Modeling Language for Stream Cipher," *Cybernetics and Information Technologies*, vol. 23, no. 2, pp. 54–71, Jun. 2023, Doi: doi:10.2478/cait-2023-0013.
- [12] W. I. Yaseen and M. F. Hassan, "Construction of Atmospheric Earth Modeling Using C++ Language," *Iraqi Journal of Science*, vol. 64, no. 5, pp. 2601–2613, May 2023.
- [13] W. Chen, Q. Yang, Z. Jiang, J. Xing, Q. Zhao, Q. Zhou, and D. Han, "Touch: A Textual Programming Language for Developing APPs of Insect Intelligent Building," *Hindawi Scientific Programming*, vol. 2020, pp. 1–26, Sep. 2020, Doi: 10.1155/2020/8887588.
- [14] V. KK Nair, T. Rayner, S. Siyambalapitiya, and B. Biedermann, "Domain-general cognitive control and domain-specific language control in bilingual aphasia: A systematic quantitative literature review," *Journal of Neurolinguistics*, vol. 60, p. 101021, Nov. 2021, doi:10.1016/j.jneuroling.2021.101021.
- [15] L. Shen, X. Chen, R. Liu, H. Wang, and G. Ji, "Domain-Specific Language Techniques for Visual Computing: A Comprehensive Study," *Archives of Computational Methods in Engineering*, vol. 28, no. 4, pp. 3113–3134, Oct. 2020, Doi: [10.1007/s11831-020-09492-4](https://doi.org/10.1007/s11831-020-09492-4).
- [16] C.-Y. Tsai, "Improving students' understanding of basic programming concepts through visual programming language: The role of self-efficacy," *Computers in Human Behavior*, vol. 95, pp. 224–232, Jun. 2019.
- [17] D. Méndez-Acuña, J. Galindo, T. Degueule, Benoit Combemale, and B. Baudry, "Leveraging Software Product Lines Engineering in the development of external DSLs: A systematic literature review," *Computer Languages, Systems & Structures*, vol. 46, pp. 206–235, Nov. 2016. Doi: [10.1016/j.cl.2016.09.004](https://doi.org/10.1016/j.cl.2016.09.004).
- [18] S. Arslan and G. Kardas, "DSML4DT: A domain-specific modeling language for device tree software," *Computers in Industry*, vol. 115, p. 103179, Feb. 2020. Doi: [10.1016/j.compind.2019.103179](https://doi.org/10.1016/j.compind.2019.103179).
- [19] N. Nikolov, Y. D. Dessalk, A. Q. Khan, A. Soylyu, M. Matskin, A. H. Payberah, and Roman "Conceptualization and scalable execution of big data workflows using domain-specific languages and software containers," *Internet of Things*, vol. 16, p. 100440, Aug. 2021. Doi: [10.1016/j.iot.2021.100440](https://doi.org/10.1016/j.iot.2021.100440).
- [20] G. Sebastián, R. Tesoriero, and J. A. Gallud, "A domain specific language notation for a language learning activity generation tool," *Multimed Tools Appl*, vol. 80, pp. 36275–36304, Sep. 2021.
- [21] S. Arslan and G. Kardas, "DSML4DT: A domain-specific modeling language for device tree software," *Computers in Industry*, vol. 115, p. 103179, Feb. 2020.
- [22] M. Vještica, V. Dimitrieski, M. Pisarić, S. Kordić, S. Ristić, and I. Luković, "Multi-level production process modeling language," *Journal of Computer Languages*, vol. 66, p. 101053, Oct. 2021.

- [23] M. Vidal, T. Massoni, and F. Ramalho, "A domain-specific language for verifying software requirement constraints," *Science of Computer Programming*, vol. 197, p. 102509, Oct. 2020.
- [24] A. Pajić Simović, S. Babarogić, O. Pantelić, and S. Krstović, "Towards a Domain-Specific Modeling Language for Extracting Event Logs from ERP Systems," *Applied Sciences*, vol. 11, no. 12, p. 5476, Jun. 2021, Doi: [10.3390/app11125476](https://doi.org/10.3390/app11125476).
- [25] B. A. Hameedi, A. A. Hattab, and M. M. Laftah, "A Pseudo-Random Number Generator Based on New Hybrid LFSR and LCG Algorithm," *Iraqi Journal of Science*, vol. 63, no. 5, pp. 2230–2242, May 2022.
- [26] J. D. KIPER, E. HOWARD, and C. AMES, "Criteria for Evaluation of Visual Programming Languages," *Journal of Visual Languages & Computing*, vol. 8, no. 2, pp. 175–192, Apr. 1997, Doi: [10.1006/jvlc.1996.0034](https://doi.org/10.1006/jvlc.1996.0034).