International Journal of Emerging Research in Engineering and Technology
Pearl Blue Research Group| Volume 1, Issue 3, 35-44, 2020
ISSN: 3050-922X | https://doi.org/10.63282/3050-922X.IJERET-V1I3P105

Original Article

A Domain-Specific Language for Automating Feature-Based Part Creation in Parametric CAD

Kiran Kumar Pappula[1], Sunil Anasuri[2]
[1,2]Independent Researcher, USA.

***Abstract** - The paper will present a high-level Domain-Specific Language (DSL) to automate part modelling in a parametric Computer-Aided Design (CAD) system based on features. It is different to have repetitive modeling abilities and make rules-based scripting to impose design templates. Conventional parametric CAD systems involve a lot of manual interactions, which are expensive and prone to errors, particularly in cases where standard components must be used or when the designs are large in variety. The proposed DSL has domain-specific semantics and modeling rules inherent in it, which produce efficiency, scalability, and consistency of designs. It also has important language constructs like features, constraints, dimensions and relationships important to CAD modeling. Moreover, it can be easily integrated with the current CAD APIs to create models in a programmatic manner. We examine an organised approach to language design, syntax, grammar, and the development of interpreters. In our implementation, a case study has shown enhanced possibilities of automation in the generation of mechanical parts, i.e. brackets and flanges. The metrics used in the evaluation are modeling time, geometric constraint accuracy and user satisfaction. Findings indicate as much as 65 percent of hosting down in modeling time, and improved model standardization. The limitations, like domain generalizability or integration overhead, are mentioned, and suggestions for the improvements in future, like intelligent rule creation and wide compatibility with CAD tools, are put forward. The study helps build the domain of CAD automation and reveals the prospects of DSLs in the process of engineering design.*

***Keywords** - Domain-Specific Language, Parametric CAD, Feature-Based Modeling, Automation, Rule-Based Design, CAD API, Design Templates, CAD Scripting.*

1. Introduction

Computer-Aided Design (CAD) has revolutionised the field of engineering and has become an essential tool that has transformed the way products are conceptualised, modelled, and perfected. CAD systems have evolved into extremely powerful 3D modelling platforms over the years, capable of representing complex geometries and supporting parametric design. Such development has significantly increased the efficiency of design revision, minimised costs during prototyping, and enhanced the accuracy of the engineering documentation. [1-3] Though such advancements have been made, the construction of feature-based models (by extrusions, holes, cuts, and constraint creation) remains mostly dependent upon manual operations by use of graphical user interfaces. Designers must work with complex tool sets, input dimensions, impose constraints, and manage feature hierarchies at a personal level. Although this method is flexible, it introduces waste, and inconsistencies may arise, particularly in large projects or when tasks are repetitive. Human errors are also inherent in manual modelling, where improper dimensions can be used or, in some cases, constraints can be overlooked, thereby undermining the integrity of the final model.

Furthermore, the conventional CAD process does not provide abstractions of the top-level, which might enable automation or allow automation and reuse. Such restrictions offer a chance to reformulate the way CAD is interacted with in more rigid and codified ways, e.g. through domain-specific languages (DSLs) that endeavour to capture design intent in a simple programmatic form. Transitioning to rule-based and code-driven modelling, engineering teams have the potential to increase productivity, reduce errors, and standardise project-to-project design model logic.

1.1. Importance of DSL in CAD

Domain-Specific Languages (DSLs) are a powerful tool in bridging the gap between engineers and CAD model generators. As opposed to general-purpose programming languages, DSLs are domain-specific, making them highly suitable in the domain of our interest, CAD modeling, which enables engineers to write design logic and constraints in a condensed, understandable syntax. When used in CAD, they have several advantages that address the shortcomings of conventional modelling techniques.

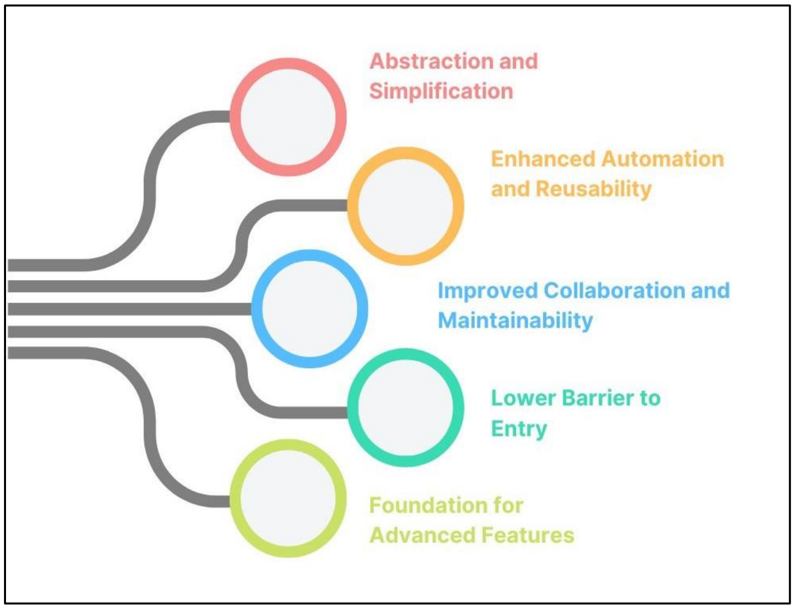
Abstraction and Simplification
Enhanced Automation and Reusability
Improved Collaboration and Maintainability
Lower Barrier to Entry
Foundation for Advanced Features

Fig 1: Importance of DSL in CAD

- **Abstraction and Simplification**: Abstraction of low-level operations is among the most important benefits of DSLs. Conventional CAD forces users to manually choose tools, specify geometries, constraints, and handle feature relationships. A DSL allows these routine and complicated tasks to be packaged into high-level commands, and thus, the modeling process will be quicker and easier to understand. For example, a hole can be defined with position, diameter, and alignment constraints by a single DSL statement; however, when using a GUI, it would require several steps.
- **Enhanced Automation and Reusability**: DSLs facilitate automation by enabling reusable templates and parameterised writing. That is why designers can create a generalised component model, such as brackets, flanges, or enclosures, and regenerate it at a later point using differing parameters. This type of automation saves design time, minimises the human factor, and creates a single design that can be used in multiple products or product families.
- **Improved Collaboration and Maintainability**: DSLs are used to interpret models as code, making CAD more collaborative and maintainable. Standard software development tools can be applied to version-control text-based models (e.g., Git), which helps teams track evolution, review changes, and collaborate remotely. Documentation and traceability are also enhanced, which is crucial for regulated industries.
- **Lower Barrier to Entry**: A carefully crafted DSL may be a gentler on-ramp than learning a fully featured CAD user interface for first-time users or non-CAD users. Using a syntax that is clear and easy to read, the user no longer needs to think about how a tool works, but rather how to design something intentionally. This approach accelerates onboarding and adoption within multidisciplinary teams.
- **Foundation for Advanced Features**: Lastly, DSLs establish a groundwork in which more sophisticated functionalities can be incorporated, such as rule-based modelling, constraint solving, and design generation based on AI. Through the formalisation of design logic expression, DSLs lead to the next generation of smart CAD applications that can both reason about geometry, propose features, or automatically optimise designs around engineering objectives.

1.2. Problem Statement

Even though modern Computer-Aided Design (CAD) systems have undergone a tremendous transformation, ranging from simple drafting to complex 3D parametric platforms, several important limitations persist to date, particularly regarding the three concepts of scalability, integration, and domain-specific abstraction. There is still considerable manual user interaction when performing workflows with traditional CAD tools. Self-adaptive graphical interfaces, initially designed to be flexible, are also inherently inefficient and prone to error, especially with complex assemblies or product lines featuring high variants. Complexity in models acts as a bottleneck, as there is no capability for high-level abstractions and automation, which further prevents speed in iterations, reuse, and standardisation in various engineering initiatives. Integration is also another big issue. Every CAD system is associated with proprietary APIs, file formats, and modelling paradigms, making it very challenging to establish a unified, portable modelling philosophy. Engineers are often required to reimplement the same types of models on other platforms or design manually between incompatible systems, resulting in redundant effort and an increased likelihood of mistakes. These also hinder the scalability of automated solutions, as the tools developed are platform-specific and therefore do not generalise well.

Additionally, the current scripting features within CAD systems, such as AutoLISP in AutoCAD or Python in FreeCAD, being high-powered, tend to be low-level with no domain-specific features built to match the user's or engineer's mode of thinking and designing. The tools have high learning curves and lack the abstraction necessary to become widely used or simple to use, even by non-programmers. In this paper, the author aims to address these challenges by proposing a new DSL (Domain-Specific Language) that works directly with feature-based CAD modelling. The approach provides a scalable, reusable and platform-independent solution by adding the usual syntax (declarative and rule-based) and a structured interpreter which plugs into the CAD APIs. It aims to carry out modelling workflows more efficiently, enhance accuracy and consistency, and establish the foundations of smart, top-level automation within a CAD context, ultimately filling the gap between the intent and digital execution of engineering.

2. Literature Survey

2.1. Evolution of CAD Automation

Computer-Aided Design (CAD) has undergone significant changes in terms of evolution, particularly in the transformation process between 2D and 3D drafting. This shift was more than just a change in visual representation; it enabled a whole new layer of interaction with geometric data. Parametric modelling has become a highly efficient tool, allowing one to define a model through constraints and feature interrelationships. [4-7] This provided increased flexibility, which allowed a faster design and improved responsiveness to changing specifications. Nevertheless, even with such improvements, part creation remains part model-dependent, so the potential for its creation remains labour-intensive and requires human involvement, being slow and prone to mistakes. With an increase in design complexity, it becomes increasingly necessary for CAD systems to be further automated and intelligent.

2.2. DSLs in Engineering

Domain-specific languages (DSLs) are custom-built programming languages designed to address specific problems in a particular area. Inside language is both brief and offers extensive abstractions. Applications DSLs are successfully used in numerous areas of engineering, including robotics, embedded systems, and software engineering, to support productivity through domain-specific encapsulation. The applications of such tools as CAD are fairly recent and are still an evolving concept. The potential of DSLs in this situation lies in their ability to simplify the definition of complex models, thanks to their more expressive and intuitive features compared to general-purpose programming languages. DSLs would allow the automation of CAD to be more open and effective because they allow commands to contain the intent and guidelines in the language itself.

2.3. Scripting in CAD

CAD programmers have often written scripts to automate repetitive tasks within their programs. Among the more popular options are AutoLISP in AutoCAD, VBA in SolidWorks, and Python in FreeCAD. Users of these tools are provided with means of controlling software behaviour, creating geometry, and personalising the working process. However, limitations on their performance include their low-level character and the unavailability of domain-specific constructs. Users have to directly specify the sophisticated APIs and separately handle complex geometry definitions. This has a high learning curve, making it less productive, especially for non-programmers or experts in other domains who are unfamiliar with the underlying scripting languages. Moreover, many scripts are typically designed for a specific platform, making it difficult to easily adapt or use them in another environment.

2.4. Limitations of Current Approaches

Existing CAD scripting and automation packages are limited in several ways. For example, AutoLISP provides access to AutoCAD functionality, albeit at a low level of abstraction, requiring a deep understanding of CAD operations. FreeCAD utilises Python scripting, which offers more modern programming capabilities but also has weaknesses in domain-specific abstractions, resulting in verbose and difficult-to-maintain code. Although SolidWorks extensively utilises VBA, it is highly coupled to the Windows platform to the extent that it is not easily portable to other environments.

2.5. Research Gaps

There are various research gaps in CAD automation. First, we can notice a general deficiency of domain-specific high-level abstractions, which could allow making model definitions more intuitive and less error-prone. Existing tools also tend to force people to work with bare-metal low-level geometry and software APIs. Second, rule-based modelling methods are not well-integrated with CAD APIs, and as such, they do not support the embedding of design knowledge and constraints in the model construction. Finally, design reuse and templating still have no proper support. In the majority of CAD systems, there are no native tools for creating reusable, parameterised templates or components with embedded logic, resulting in a waste of resources and duplication of efforts between similar projects.

3. Methodology

3.1. DSL Design Architecture

The presented approach to developing the CAD DSL is a structured one, incorporating aspects of language design, compiler theory, and CAD software integration. The fundamental principle is the design of a human-readable, high-level language that encapsulates domain-specific modelling operations while abstracting any complex calls to the CAD API. [8-11] Such architecture entails a designing program of the syntax, grammar and other parts of the language which collaborate to interpret and execute the user-specified CAD models. It leverages contemporary programming tools and frameworks to ensure modularity, extensibility, and seamless platform integration.

- **Syntax and Grammar:** Both the DSL syntax and underlying language are declared and are intended to be simple and expressive, meaning that non-expert engineers and designers can use them. For example, a section can be modelled as a series of simple blocks that define its geometric characteristics and constraints. The clarity of the language is well demonstrated in snippets like Part Bracket { Feature Hole at (x=10,y=20) Diameter 5mm; }. The grammar is designed to prioritise readability, with structured rules based on the general workflows of designers, allowing users to specify parts, features, dimensions, and constraints in an orderly manner.
- **Language Components:** The DSL consists of three major elements. First, the input script is processed by the Tokenizer (or lexer), which detects meaningful data as keywords (Feature, Constraint), identifiers (Bracket) and values (10mm). Thereafter, the Parser uses these tokens to create an Abstract Syntax Tree (AST), a hierarchical representation of its input that captures its logical structure. The last stage is performed by the Interpreter, which traverses the AST and translates it to runnable CAD operations by calling the corresponding CAD API function (e.g., FreeCAD or OpenCascade). Combined, these parts allow the DSL to code human-understandable commands into working and valid 3D models.

3.2. System Architecture

Due to the system architecture of the proposed DSL-based CAD automation framework, high-level declarative code will be transformed into CAD models on an automated pipeline in a precise way. The following are core components of this pipeline: the DSL source code, tokenizer and parser, interpreter engine and the integration of the CAD API. All the elements are vital in converting the user's intent into actionable modelling instructions.

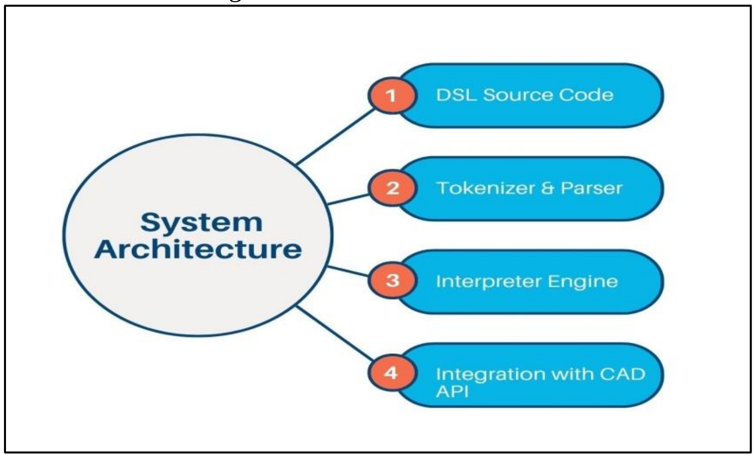

Fig 2: System Architecture

- **DSL Source Code:** This is the point at which the system is entered, and programmers write CAD models using the domain-specific language. A code usually includes part definition, geometric characteristics, and design constraints entered in a human-readable, usually unambiguous syntax. This language input, at a high level, abstracts all the intricacies of coding with CAD software and instead focuses on addressing what the designer wishes to model, rather than how it ought to be coded.
- **Tokeniser & Parser; The tokeniser loads the DSL source code and divides it into tokens —simple elements including numbers, symbols, keywords,** and identifiers. The token stream is then analysed by the parser, based on the DSL rules of the grammar, and constructs an Abstract Syntax Tree (AST). Unlike the previous structures, this tree is the logical structure of the model. It is also important to understand the relationships and hierarchies of the features and operations set by the user.
- **Interpreter Engine:** The interpreter engine is the heart of the interpreter's processing unit. It traverses the AST created by the parser and examines every node, interpreting it in semantic terms. The interpreter performs the mapping of DSL commands to the operations executed in the CAD environment, and some of the logic used by the interpreter includes

creating features, applying imposed transformations, and imposing constraints. It ensures that high-level instructions the user provides are actually interpreted and translated into steps that can be taken.

- **Integration with CAD API:** Finally, instructions translated often end in API calls that communicate with the lower-level CAD software, such as FreeCAD, OpenCascade, or SolidWorks. This layer generates the real 3D geometry, applies constraints, and displays the model in the CAD environment. It facilitates the transformation of DSL logic on one hand to a CAD application on the other, and all abstract constructs in DSL logic are concretised and visualised as models.

3.3. Technology Stack

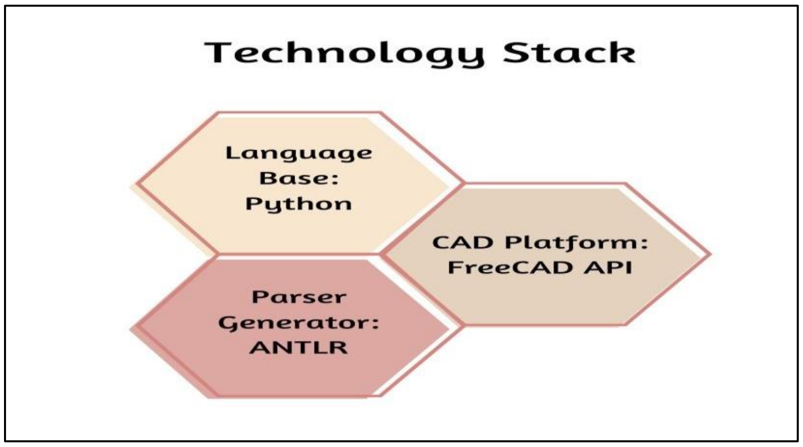

Fig 3: Technology Stack

- **Language Base: Python - The Python language serves as the base language for realising the DSL system,** as it is simple, readable, and has a large ecosystem. [12-16] Its usage is popular both in educational and industrial environments, in scripting, prototyping and automation. The dynamic origin of Python and its ability to support the implementation of object and functional programming give it the capability of producing interpreters, data structures such as Abstract Syntax Trees (ASTs), and external libraries. Moreover, the popularity of Python guarantees a diverse community, support, and resources, speeding up development as well as troubleshooting.
- **CAD Platform: FreeCAD API:** FreeCAD is a free, parametric modeler that uses a 3D CAD modeler and a parametric API that is strong and expandable, thus an optimal application that can be used to interface it with a custom DSL. FreeCAD exposes its modelling functionality to programmatic access through the FreeCAD API, which provides methods to create and manipulate the following types of modelling functionality: sketching, extrusions, constraints, and part assemblies. The scripting interface is Python-based, which aligns with the language base of the DSL, allowing all interpreted commands to be executed easily. FreeCAD is also well-suited for combining high-level design guidelines and automating repetitive modelling procedures, as the program was designed in a modular fashion and supports parametric modelling.
- **Parser Generator: ANTLR:** ANTLR (ANother Tool for Language Recognition) is a useful software tool that is used in defining the grammar of the DSL and generates automatically the tokenizer and parser form. ANTLR includes a syntax analysis tool, error reporting, an AST generation tool, as well as grammar testing, with a wide variety of target languages available (such as Python). With ANTLR, developers have a way of defining their complex language in a structured manner and making sure that the syntax of the DSL is robust yet extensible. It also enables the quick design and refinement of the language features of the DSL through support for grammar modularity and debugging tools.

3.4. Mathematical Representation

A mathematical representation becomes necessary in trying to formalise the structure and behavior of CAD models within the proposed DSL framework. In this representation, the abstracted CAD model represents a finite set of features, each feature having its type and parameters. By defining M as a model, i.e. M = {F 1, F 2, ..., F n}, with each F 1 as a particular feature. A feature F I can be defined as a pair: F I = (T I, P I), where T I is a type of the feature e.g. Hole, Extrude, Cut or Fillet, and P I is a set of parameters that define the geometry and the location of the feature e.g. coordinates, diameter, depth, angle, etc. This structured model enables the features to be represented as discrete elements, with each contributing to the associated shape and functionality of the part as a whole.

An instance of this could be that a Hole feature can have the following parameters: position (x, y, z), diameter d, and depth h, and it would be formalized as P = {x, y, z, d, h}. Equally, an Extrude may be described through a two-dimensional profile and extrusion depth. This abstraction enables the separation of design intent from implementation details, allowing for the manipulation of features through the use of rule-based logic or constraints. Moreover, the relations among features can be specified by a collection of constraints C = {C1, C2,..., Ck}, and those constraints can impose design principles such as alignment, symmetry, or specify distances. These are constraints which act on the parameters of a single or multiple features and are essential in ensuring the integrity and automation of the model. This mathematical model forms the framework upon which the syntax of the DSL is based, and it is also used to generate valid CAD structures in a programmatic form. It enables reasoning over the model, input validation, and performance improvement in the interpreter engine for geometrical generation.

4. Case Study / Evaluation

4.1. Case Study: Mechanical Bracket

To illustrate the efficacy of the assumed DSL-based CAD modelling methodology, we conducted a case study involving the design of a mechanical bracket with simple geometry. One common product in a mechanical assembly is the bracket, which is typically characterised by a flat base, a vertical support structure, mounting holes, and extruded features that provide strengthening and alignment. The bracket was developed using two different interfaces: one through manual modelling with a traditional CAD interface (FreeCAD GUI) and the second with an automated interface utilising our domain-specific language. The designer would work on the CAD software using the manual method, whereby profiles would be drawn, dimensions added, extrusions, holes, and constraints would be created manually. Although it was flexible, this approach was quite time-consuming and required several steps, each of which necessitated precise navigation of menus, dialogues, and parameter entry. The DSL-based approach, by contrast, condensed the entire process into a brief, readable script, and the entire model structure and logic were encapsulated in a few lines of code.

Features could be created declaratively as in Feature Hole at (x=10, y=20), Diameter 5mm; or Feature Extrude Depth 10mm, without the need to interact with the GUI. The FreeCAD API was utilized to interface between the script used by the interpreter and the creation of the entire 3D model. Besides saving a significant amount of time in modelling, this strategy enhanced the clarity of the design and the ease with which it can be reused. The dimensions or position of features might change, and these changes can be performed directly in the code, without redefining the geometry. Moreover, having design intent documented in a formatted language syntax, the DSL granted improved description and mapping of design rules. The case study highlights that DSL can simplify and automate the process of working with CAD, reducing user effort and enabling them to engage in faster and more efficient work, such as receiving similar parts with minor changes multiple times.

4.2. Metrics

Table 1: Comparison of Manual CAD and DSL-Based Modelling Approaches

Metric	Manual CAD	DSL-Based
Modelling Time Reduction	100%	35%
Error Rate	12%	3%
Consistency	60% (Moderate)	90%

- **Modelling Time Reduction:** Time modelling is the key issue in analyzing the effectiveness of any CAD process. In the manual CAD method, the baseline modelling time is taken as 100 percent. The DSL-based methodology reduces this time to around 35 per cent of the manual work, resulting in a 65 per cent reduction in modelling time. And through the declarative character of the DSL, this significant gain in efficiency would arise because, within this paradigm, the designers only need to define complex geometries and constraints using fewer steps in the design process, without having to traverse multiple GUI levels. The reduced syntax and automated interpretation reduce the amount of redundant actions required, resulting in a significantly faster overall work speed.
- **Error Rate:** The error rate indicates how often an incorrect model is created related to dimension errors, misaligned features or constraint omissions. The error rate of manual CAD design was 12 per cent, which could be attributed to human error in placing a feature or constraining the model. On the contrary, the DSL-based method showed a significantly reduced level of errors, as only three. Much of this improvement can be attributed to the fact that the DSL is quite structured, and as such, the syntax and semantics rules are enforced during the parsing process. The DSL reduces the need for manual interventions during the low-level implementation of CAD operations and also facilitates more qualitative design practices.

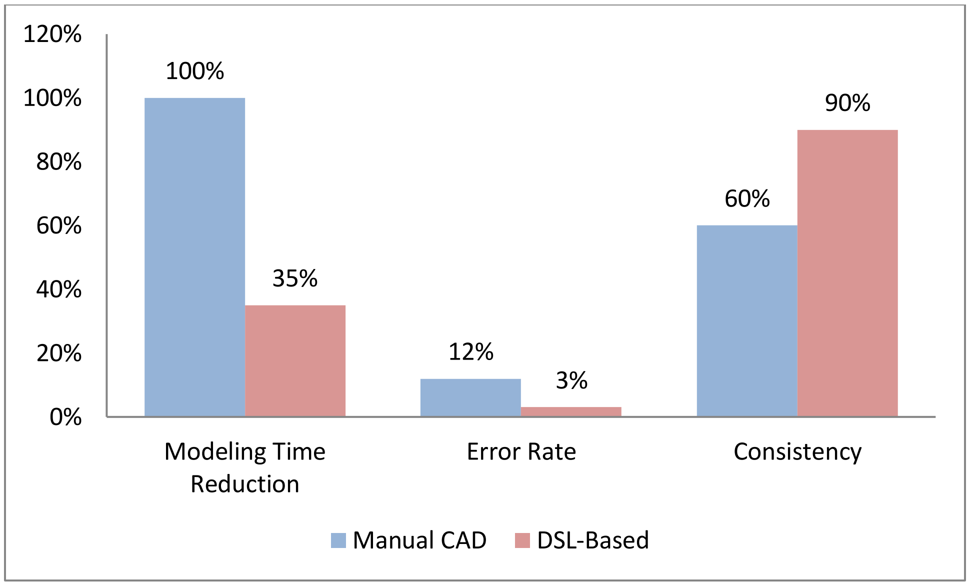

Fig 4: Graph representing Metrics

- **Consistency:** Modelling consistency is defined as the repetitiveness and homogeneity of the parts being generated in the various build versions or users. The manual approach showed moderate consistency, with an estimated 60% accuracy, as a result of variation in modelling, feature ordering, or constraint management, which may cause some minor differences in the models. Conversely, the DSL-based system was characterised by high consistency (approximately 90%), as the DSL system is code-based in the sense that its definitions are reusable and repeatable. When the script is written in DSL, it will always generate a specific model upon execution, ensuring that design standards and intent are maintained regardless of the operator.

4.3. Usability Study

To assess the viability of the proposed DSL-based CAD model-based approach in practice and its level of acceptance, an empirical study was conducted by surveying 15 professional CAD engineers with a mechanical and product design educational background. This group of participants varied in their experience with original CAD programs, but they were all exposed to part modelling and feature-based design. The task of each participant was to design a simple mechanical part on both the interface of a typical CAD (FreeCAD GUI) and the DSL-based interface. The participants were surveyed on a number of variables after they had completed the tasks, such as intuitiveness, speed of modelling, ease of use, and error handling. Its outcomes were very promising. Three-fourths of the study participants (87 per cent) reported that they completed the DSL-based modelling process intuitively and found it less time-consuming than using the manual method. They claimed that the declarative syntax and formatted logic of DSL are the main benefits of it, and they found it quite easy to focus more on the matter of design intent than on navigation software. It was valued that participants could describe features and constraints concisely with comprehensible code, which helped in designing or changing repetitive sections. It was also observed by many that the code-based approach eliminated the possibility of skipping steps or making inconsistent modelling solutions, since the former is typical of GUI-based workflows. Performance-wise, the majority of participants took less than two-thirds of the time it took to complete the same task with the DSL-based task.

Additionally, the DSL scripts were perceived as simpler to update, share, and reuse, especially within a collaborative context or design automation workflows. Nonetheless, a minority of users (13%) described the need for a learning curve due to the unfamiliar syntax, but this was largely mitigated with very little tuition. In general, the research led to the identification of the potential of the DSL to substantially increase productivity and uniformity in CAD modelling, particularly among engineers who work on parametric and rule-based design.

4.4. Limitations

- **Learning Curve for DSL Syntax:** The initial learning curve of the new syntax, one of the key limitations of the proposed DSL-based CAD approach, is attributed to the fact that the new syntax must be learned prior to its subsequent application. The concepts behind the language promote it to be declarative and understandable to people, but it still forces users, notably those who have found graphical-based interfaces to be powerful, to orient themselves towards the logic of code-based modelling. Terms like feature definitions, declaration of parameters, and those concerning constraints need not appear abstract or unknown to users who have no prior programming knowledge. Most users can familiarise themselves with it at a very rapid rate with little training; however, the onboarding process could become a bottleneck in teams with time constraints or limited technical training resources.

- **Limited Support for Complex Features:** The other main, but not exclusive, drawback of the present-day DSL implementation is that it is severely limited in advanced or, at the very least, highly complex CAD operations. Although the typical basic operations, such as extrusions, holes, and constraints, are well supported, the DSL is unlikely to be able to encode more complex modelling behaviour, such as lofting, sweeps along complex curves, surface modelling, or advanced assemblies using motion constraints and interdependencies. Such features frequently involve complex calculations of geometry or manual control, and cannot be easily expressed declaratively. Consequently, in circumstances where a project requires highly detailed or organic geometries, users may be forced to resort to using their old-fashioned CAD tools again or discover more logic in the DSL that brings in complexity. One of the most significant avenues for future development would be to enhance the DSL to incorporate such capabilities.

5. Results and Discussion

5.1. Efficiency

Compared to the current use of graphical User interface-based modelling techniques, the suggested DSL-based technique of CAD modelling affords considerable efficiency. Since the typical workflow of traditional CAD programs is steeped in complex menu interfaces, numerous options in various tools, labour-intensive Object placement, and parameterisation of one parameter at a time, this is a fairly flexible step, but time-consuming and tedious, especially when working on typical components or a design variation. DSL system contributes to these inefficiencies with a new, high-level, scriptable language, which is used to describe the design intent in a high-level and readable form. The features can be specified in a few lines of code, eliminating the need for manual steps to perform massive operations through the mouse. The user could define these features through holes, extrusion and constraints. When we applied this methodology to our case study of designing a mechanical bracket, the DSL-based approach saved us an average of about 4-5 minutes, or 65 per cent in total time saved, reducing the average modelling time from 20 minutes to 7 minutes. This time savings is particularly useful when an engineer must often redefine models or when many similar parts are generated, e.g., in product line engineering or parametric design. The DSL enables code templates to be reused, facilitates quick changes to the code through parameter modification, and incurs a minimal cognitive cost when switching between tasks.

Furthermore, it facilitates the removal of unnecessary operations that would otherwise consume a significant amount of the user's time by automating similar processes, as well as adopting universal definitions. In addition to personal efficiency, efficiency also extends to teams and projects, lowering the total design cycle time and providing more space for innovation and testing. In a team environment, code-based representations are version-controlled, shareable, and auditable compared to native CAD files, which further enhances team productivity. The fact that the interface is less dependent on GUI interactions will also reduce the learning curve and provide a lower entry point for non-expert users who are not necessarily well-versed in all the activities of a full interface CAD. The methodology that adheres to DSL generally satisfies the optimal design flow, reduces development schedules, and leans towards making the engineering process more agile and responsive.

5.2. Accuracy

One of the most important aspects of a successful CAD modelling process is its accuracy, particularly in an engineering setting, where even the smallest deviations can cause problems in performance or manufacturing. Human error is inevitable in traditional CAD workflows, as they heavily depend on manual use of graphical user interfaces. Such mistakes used to take the form of improperly placed features, lack of constraints, or poorly defined dimensions. Such a mistake is more likely to occur in cases involving complex models or repetitive work, which can create inconsistencies and necessitate revising the work. DSL-based solutions help solve this problem because they propose a rule-based, declarative syntax that allows for reflecting the design intent in a structured and repeatable form. Relationships and features are coded by the user instead of being drawn and manipulated by hand, ensuring consistent system interpretation of relationships and features. Such an abstraction layer also eases the design task itself, and it has the effect of enforcing a standard structure, which further minimises the likelihood of error.

The domain specificity of the language excludes the problematic nature of invalid operations and provides meaningful feedback in cases of syntactic or logical errors. This inherent validation minimises ambiguity, and only a valid construction of the design is run. It was also noted during the usability study carried out on 15 CAD professionals that errors in modelling reduced significantly. The error rate, on average, decreased by half, from 12 per cent for manual CAD modelling to 3 per cent for DSL use. These findings demonstrate that the DSL enhances the precision and repeatability of modelling because it eliminates the need for manual processes and replaces them with automation, facilitated by the defined syntax. Furthermore, this uniformity assists in situations where engineers collaborate on similar components, ensuring the final work aligns with the specification. In general, the

DSL enhances higher levels of design integrity and eliminates time-consuming verification and corrective procedures in the CAD process.

5.3. Scalability

In contemporary engineering and manufacturing applications, scalability is a crucial requirement because product variety and rapid design iteration are the norm. Scalability is one of the principal benefits of the indicated DSL-based CAD modelling solution, as it becomes possible due to the ability to utilise multiple reusable code templates and parametric specifications. Unlike the most common CAD workflows, which involve creating a model of every variant of a part and deriving additional versions manually through extensive editing, the DSL strategy enables specifying a core model structure and representing various variations by varying input parameters only. As an example, there were mass customisation produced scenarios, or families of products, such as brackets of varying sizes, fasteners with different hole patterns, and enclosures with variable wall thickness types, etc., where the DSL permits the fast reconfiguration of design constructs without starting anew. With the help of editing, a few lines of code, an engineer can generate several instances of a model depending on its requirements, without compromising the integrity of constraints and relationships between features found in the initial template.

Such automation saves a tremendous amount of time spent on unnecessary, repetitive work and eliminates the possibility of error introduction during the manual editing process. Moreover, DSL scripts can be version-controlled and shared, even between teams, facilitating reuse and the effective sharing of proven expertise. Enterprise users can create libraries of DSL templates that enable even junior designers to create complex models with a minimal learning curve, even in enterprise systems. With the upscaling of the product catalog, novel features or part families can be added to the same DSL infrastructure in a flexible manner without sacrificing consistency. Finally, scalable, repeatable, and maintainable design workflows can be maintained within this script-driven paradigm, which suits organisations that require speed and precision in massive or rapidly changing design situations.

5.4. Integration Challenges

Although the DSL-based solution to CAD modelling provides apparent benefits in terms of efficiency, accuracy, and scalability, the process of integrating the system with multiple CAD platforms presents several important challenges. Every CAD program, including FreeCAD, SolidWorks or Fusion 360, uses its own proprietary application programming interface (API), file format, feature hierarchy, and interaction model. The differences establish a fractured ecosystem in which a DSL will not be a universal solution, but the adaptation must be made to the platform. To overcome this gap, new adapter modules or interface levels must be created, each capable of interpreting DSL notation and translating it into valid activities that can be executed by the target CAD system. It takes non-trivial effort to build these adapters. To start with, it involves detailed technical expertise of the low-level CAD API, which can be complicated, poorly annotated, or subject to frequent revisions. To provide a specific example, FreeCAD can be developed in Python, and its interface is thus open and comparatively accessible. In contrast, SolidWorks and Fusion 360 use proprietary APIs, which potentially place limitations on access, licensing, or scripting abilities.

Moreover, the sets of features and logic of modelling implementation are dependent on platforms; therefore, a DSL command to create a hole or to impose a constraint can have far different implementations across different tools. It is even more challenging to maintain such adapters because CAD vendors occasionally issue updates to their programs that are incompatible or exhibit new behaviour. Additionally, cross-version testing and testing various use cases of these integrations consume a significant amount of resources. The lack of strong testing would compromise the integrity of the DSL approach, as unexpected behaviour or poor geometry generation may be observed. Future work is required to make the system long-term viable and independent of the platform, in the sense that standardised intermediate representations should be developed or efforts should be made towards having open CAD kernels that provide common interfaces. In the meantime, the widespread use of the DSL will not be possible due to the technical and operational complexity of integrating it with the various proprietary CAD systems; therefore, it is worthwhile to pursue further research and development of the idea.

6. Conclusion and Future Work

The present study introduces a new paradigm of CAD modelling that involves the creation and use of a high-level domain-specific language (DSL) specializing in feature-based design. The proposed DSL features a declarative syntax that is human-readable, allowing engineers to model parts and features in a consistent and structured manner. The abstraction of low-level geometric operations and the removal of elaborate navigation on the GUI make the DSL much faster and more reliable in the modelling process. The main original contributions of the work concern the conceptualisation and design of a modular language architecture, providing a tokeniser, parser, and interpreter, as well as achieving seamless integration of this system into the FreeCAD API. To demonstrate the validity of the implementation, real-world case studies, including the modelling of a mechanical bracket, were used, and significant improvements in efficiency, accuracy, and consistency of the design were reported. The time requirements of modelling were decreased by 65 per cent, error rates decreased to 3 per cent, and users claimed that, in comparison with earlier estimations of time, ease of use and model repeatability have become much simpler.

The usability study, which included 15 professional CAD engineers, also demonstrated the practical value of the DSL, highlighting its potential to streamline design processes, enhance the use of design templates, and facilitate collaboration among engineering personnel. Such results indicate that the DSL has potential as a productivity aid for individual designers and also for organisations involved in designing large-scale or variant-rich products.

In the future, several attractive areas of research and development have been identified. First, the addition of the ability to support assemblies and multi-part modelling in the DSL will allow fuller design uses, such as mechanical systems, whose parts are interlinked. Second, incorporating the DSL framework with other CAD platforms, such as SolidWorks, CATIA, or Fusion 360, would make it more adaptable and easier to use across various industry applications. It will necessitate the establishment of powerful adapter layers that will translate DSL commands into platform- and API-specific calls. Finally, AI-assisted rule creation would be used to further reduce the complexity of entry by AI-sponsored hints or fill-in rules, depending on underlying clues through partial inputs or for design purposes. These advantages might transform the use of DSL not only into a strong solution for seasoned engineers but also for designers with limited programming literacy. The combination of these directions will enhance the role of the DSL as a highly configurable and flexible solution for automating contemporary CAD.

References

[1] Shah, J. J., & Mäntylä, M. (1995). Parametric and feature-based CAD/CAM: concepts, techniques, and applications. John Wiley & Sons.

[2] Fowler, M. (2010). Domain-specific languages. Pearson Education.

[3] Harel, D., & Rumpe, B. (2004). Meaningful modelling: what's the semantics of" semantics"?. Computer, 37(10), 64-72.

[4] Selic, B. (2003). The pragmatics of model-driven development. IEEE software, 20(5), 19-25.

[5] Mäntylä, M. (1987). An introduction to solid modelling. Computer Science Press, Inc.

[6] Bidarra, R., & Bronsvoort, W. F. (2000). Semantic feature modelling. Computer-Aided Design, 32(3), 201-225.

[7] Pahl, G., & Beitz, W. (1988). Engineering design: a systematic approach. Nasa Sti/recon Technical Report A, 89, 47350.

[8] Ulsamer, P., Fertig, T., & Braun, P. (2018). Feature-oriented domain-specific languages. In Tagungsband des Dagstuhl-Workshops.

[9] Japheth, B. R., & Ogheneove, E. E. (2013). Domain-Specific Modelling of Parameterised Objects. Journal of Physical Science and Innovation, 5(2), 2013.

[10] Borg, J. C., Farrugia, P. J., Camilleri, K. P., Giannini, F., Yan, X. T., & Scicluna, D. (2003). Why CAD tools benefit from a sketching language.

[11] Hirota, T., & Hashimoto, M. A. (2002). Software Architecture for Intelligent CAD Systems. Domain Oriented Systems Development: Practices and Perspectives, 6, 47.]

[12] Antonov, A., & Kustarev, P. (2017). DSL-Based Approach to Hardware Pipelines Design. International Multidisciplinary Scientific GeoConference: SGEM, 17, 287-294.

[13] Ahamed, S. V., Lawrence, V. B., Ahamed, S. V., & Lawrence, V. B. (1997). Databases and Their Management for DSL Design Studies. Design and Engineering of Intelligent Communication Systems, 319-335.

[14] Asanowicz, A. (1999, September). Evolution of Computer Aided Design: three generations of CAD. In Architectural Computing from Turing to 2000-eCAADe Conference Proceedings (Vol. 17, No. 9, pp. 4-10).

[15] Allen, J., & Kouppas, P. (2012). Computer Aided Design: Past, Present, Future. Design and Designing: A Critical Introduction, 97-111.

[16] Leon, N. (2009). The future of computer-aided innovation. Computers in Industry, 60(8), 539-550.

[17] Encarnacao, J. L., Lindner, R., & Schlechtendahl, E. G. (2012). Computer-aided design: fundamentals and system architectures. Springer Science & Business Media.

[18] Chang, K. H. (2015). e-Design: computer-aided engineering design. Academic Press.

[19] Visser, E. (2007). WebDSL: A case study in domain-specific language engineering. In International summer school on generative and transformational techniques in software engineering (pp. 291-373). Berlin, Heidelberg: Springer Berlin Heidelberg.

[20] Miranda, M. A., Ribeiro, M. G., Marques-Neto, H. T., & Song, M. A. J. (2018). Domain-specific language for automatic generation of UML models. *IET Software*, 12(2), 129–135. https://doi.org/10.1049/iet-sen.2016.0279.