Physics4All DSL: A Domain-Specific Language for Democratising Physics Simulations and Advancing DSL Engineering with JetBrains MPS

Sofia Meacham[1][a], Clément de La Bourdonnaye[2], Vaclav Pech[2], Hessa Alfraihi[3]
[1]*School of Computing & Engineering, Bournemouth University , Fern Barrow, Poole, UK*
[2]*JetBrains s.r.o., Prague, Czech Republic*
[3]*Department Of Information Systems, College of Computer and Information Sciences, Princess Nourah bint Abdulrahman University, Riyadh, Saudi Arabia*
smeacham@bournemouth.ac.uk, cle.bourdonnaye@gmail.com, vaclav.pech@jetbrains.com, haalfraihi@pnu.edu.sa

Keywords: Domain-Specific Languages; Physics Education; Simulation; JetBrains MPS; Model-Driven Engineering.

Abstract: Simulations are essential in physics education but remain difficult for non-programmers to design and adapt. Existing tools often limit customisation, hindering teachers and students from tailoring experiments to their needs. This paper presents Physics4All, a domain-specific language (DSL) built with JetBrains MPS to democratise the creation of physics simulations. Physics4All introduces domain-specific constructs—worlds, objects, forces, dimensions, and vectors—expressed in familiar mathematical notation. Its modular generation pipeline supports multiple targets (Java and JavaScript), enabling simulations to run across platforms without altering models. Key innovations include implicit unit conversion, reusable forces and objects, and live type checking for correctness. Beyond the educational domain, these features illustrate generalisable DSL engineering principles for modularity, abstraction, and reusability. We evaluated Physics4All through a metrics-based comparison with a GPL baseline and an empirical case study involving secondary school teachers and educational technology developers. The comparison highlighted substantial reductions in implementation effort, while the case study confirmed high suitability, expressiveness, and productivity, with usability and maintainability identified as areas for improvement. Compared to widely used tools such as PhET, Algodoo, and COMSOL, Physics4All offers greater customisation while remaining accessible to non-programmers. The results demonstrate how DSLs can expand the reach and impact of simulation-based education while contributing to broader discussions on domain-specific language engineering.

1 INTRODUCTION

Simulations play a central role across domains ranging from the Internet of Things (de Paula Ferreira et al., 2020) to finance (McLeish, 2011) and education (Rehman et al., 2021). In physics education, simulations can enhance engagement and conceptual understanding, yet creating or adapting them typically requires advanced programming expertise (Abar et al., 2017). As a result, teachers and students often depend on predefined environments such as PhET or Algodoo. While these tools prioritise usability, they restrict the ability to customise underlying models or equations, limiting their applicability to advanced topics or tailored experiments. This creates a gap for educators who seek both accessibility and flexibility.

We address this challenge with *Physics4All*, a domain-specific language (DSL) for physics simulations developed in JetBrains MPS. The language provides high-level abstractions—objects, worlds, forces, units, and vectors—expressed in familiar mathematical notation, while hiding technical complexities. Through MPS's generator framework, models are compiled into runnable simulations on multiple platforms (Java, JavaScript), and built-in type checking ensures correctness. Unlike existing tools, Physics4All enables educators to define custom forces and equations, extending the range of phenomena that can be modelled.

Importantly, the contribution of Physics4All extends beyond physics education. The design illustrates several DSL engineering principles of general interest: modular generator planning, reusable abstractions, integrated unit and dimension han-

[a] https://orcid.org/0000-0002-8474-4917

dling, and live type checking. These innovations balance accessibility for non-programmers with the rigour expected in model-driven engineering, making Physics4All both a practical educational tool and a case study in DSL engineering.

Contributions. This paper makes the following contributions:

- A DSL-based approach that empowers physics educators and students to design simulations without programming knowledge.
- An extensible architecture supporting reusable forces, implicit unit conversions, vector notation, and live type checking.
- A modular code generation pipeline that targets multiple runtimes while keeping models implementation-agnostic, illustrating reusable principles of DSL engineering.
- A metrics-based comparison of GPL and DSL implementations, demonstrating significant conciseness and clarity gains.
- An empirical case study with physics teachers and educational technology developers, highlighting functional suitability, expressiveness, and usability.
- A comparison with widely used simulation tools, positioning Physics4All as uniquely both accessible and customisable.

2 BACKGROUND STUDY

The background section introduces the role of physics simulations in education and the potential of domain-specific languages (DSLs) to address existing challenges. The first subsection discusses the benefits and limitations of physics simulations, while the second explores DSLs as a solution for enhanced customisation and accessibility.

2.1 Physics simulations for education

Over the past two decades, several efforts have sought to modernise physics education in secondary schools through simulation environments. In (Ogegbo and Ramnarain, 2022), the authors propose a guided-inquiry framework using interactive simulations to enhance conceptual understanding. Similarly, (Rehman et al., 2021) report successful classroom trials, though their work did not extend to large-scale deployments or broader curricula.

Other studies highlight both the promise and limitations of simulation-based approaches. (Campos et al., 2020) show how simulations can be adapted to different European contexts, while (Ben Ouahi et al., 2022) emphasise that teachers often lack training to embed them effectively. Likewise, (Çelik, 2022) underline the importance of simulations that complement rather than replace traditional methods.

Most initiatives rely on established software such as PhET (PhET Interactive Simulations, 2023; Banda and Nzabahimana, 2023) or Algodoo (Algodoo, 2023). These platforms excel in accessibility and engagement, offering interactive representations of fundamental concepts such as mechanics, optics, and waves. As a result, they succeed in motivating students and supporting inquiry-based learning.

However, three main limitations persist. First, existing tools rarely support advanced topics such as electromagnetism, relativity, or thermodynamics, despite their presence in secondary curricula. Second, teachers have little scope to adapt simulations to their own learning objectives or local curricula. Third, infrastructure and training barriers remain: while free tools like PhET are widely used, more advanced packages often involve licensing costs, and teachers frequently lack the training to integrate them effectively. Consequently, both students and teachers remain dependent on predefined, one-size-fits-all environments.

In summary, physics simulations demonstrably improve engagement and learning, but their educational impact is constrained by scope, customisability, and integration challenges. These limitations motivate the search for approaches that empower teachers and learners with more flexible tools.

2.2 Domain-specific languages as a solution

Domain-Specific Languages (DSLs) provide a promising avenue to address these challenges, particularly when customisation is required (Mernik et al., 2005). DSLs have been successfully applied in domains including finance (McLeish, 2011), automotive systems, and the Internet of Things (de Paula Ferreira et al., 2020). In these areas, abstraction and separation of domain concerns from technical implementation have enabled non-programmers to contribute meaningfully to complex systems.

Abstraction is central to the contribution of DSLs. By removing low-level programming details, DSLs act as "boundary objects" between technical developers and domain experts. In education, this allows teachers and instructional designers to express physics concepts directly, without translating them into code. For educational technology developers, DSLs provide a structured way to incorporate teacher-

defined content into platforms, ensuring alignment between pedagogical goals and technical implementation.

DSLs are not without challenges: they require careful language design to balance expressiveness and simplicity, and their long-term maintainability must be considered. Nevertheless, their potential to democratise technology and empower non-programmers makes them particularly well suited to physics education. A DSL for simulations could enable teachers to define and adapt their own experiments, extend existing ones, and better align simulation-based learning with curricular needs. This vision motivates the development of Physics4All, presented in this paper.

3 PROPOSED SOLUTION – PHYSICS4ALL DSL

To bridge the gap between physics simulations and accessibility, we introduce Physics4All, a domain-specific language (DSL) tailored for the physics domain. Designed to be intuitive, it enables educators, students, and educational technology developers to define and interact with simulations without programming expertise. The DSL integrates mathematical notation, vector operations, and rotation matrices, ensuring a natural representation of physics concepts. By leveraging MPS BaseLanguage and KernelF, Physics4All provides an extensible framework that simplifies simulation creation while retaining computational power required for complex physics experiments. The DSL is available as open source on GitHub (Vaclav, 2025).

3.1 Architecture

The primary design goal of Physics4All is to allow users to express simulations directly in physics terms, without low-level programming. Key requirements included support for standard mathematical operations (addition, multiplication, exponentiation), vectors, and rotation matrices. These abstractions reduce verbosity compared to general-purpose languages and align closely with how physics is taught.

The language architecture combines existing MPS DSLs. BaseLanguage (an MPS version of Java) provides generators, while KernelF (Völter, 2020) supplies reusable mathematical expressions. KernelF's modularity allows Physics4All to adopt only the needed components and extend them with domain-specific concepts such as forces and matrices (Tozzi, 2017; Völter, 2018).

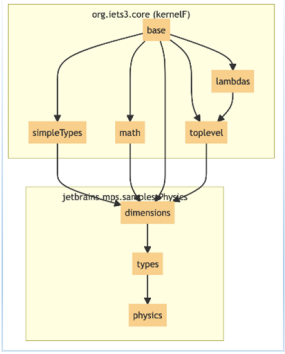

Figure 1: Language dependency graph. A → B indicates that B uses and/or exposes concepts from A.

Figure 1 illustrates the dependency structure: Physics4All builds on the *types* language, which in turn inherits from *dimensions*, itself integrating KernelF modules such as *simpleTypes* and *math*. Each language can utilise concepts from any of its ancestors in the graph. For example, Physics4All reuses type checking and mathematical operations while adding physics-specific constructs.

3.2 Language structure

The high-level DSL structure of the most important concepts is depicted in Figure 2. The language offers several high-level concepts for describing simulations. As shown in Fig. 2, the root (top-level) concept of the DSL is the *Simulation* concept. This concept manages the simulation configurations of physical objects and their behaviour within a particular world. Users can define the world to be simulated, set the camera position, adjust the simulation speed, and specify the metrics to be displayed.

Worlds contain definitions of objects and may represent parts of a larger simulation. They can also import other worlds, which can be positioned according to the user's needs. A *Simulation* can contain a one-to-many (1:*) relationship with *WorldDefinition*, which in turn consists of a one-to-many (1:*) relationship with *ObjectDefinition*.

Simple physical objects are represented by the *ObjectDefinition* concept. Users can set initial parameters for these entities—such as size, shape, velocity, position, and forces—which are then reflected in the simulation. An *Object* can have a *StyleDefinition*, which includes properties like *shape*, *collision behaviour*, *texture*, and *trace effects*, as well as an associated *Force* concept.

The DSL allows forces to influence an object's motion over time. A *Force* can be Conditional, Moment, or Simple. A Simple force is further categorised as either Static (computed once and applied

Figure 2: High-level DSL concepts - architecture

constantly) or Dynamic (computed at each simulation step).

Abstract concepts are used in multiple places to promote reusability and improve the overall language design (Voelter, 2017).

For example, it is useful to define a type of force that can be reused without needing to redefine it multiple times. Custom forces enable users to define such forces and provide parameters for customisation. For instance, we can define an interaction force as a force acting toward or away from surrounding objects, with the force's magnitude as a configurable parameter. Using this, we can define gravitation as an interaction force that depends on the mass and distance between objects. Finally, this force can be applied to any object in the simulation.

This demonstrates that the Physics4all DSL offers extensive customisation options, allowing users to define underlying physics equations and algorithms.

Another example is that each object can inherit from an abstract object, which, similar to custom forces, allows for defining default properties for derived objects. For instance, we can define a custom object Planet that uses the gravitational force described above and has a spherical shape with a radius dependent on mass. Then, we can create objects Earth and Moon, both inheriting from Planet. If the default properties assigned to Planet do not fully meet our needs, we can override them for individual objects. For example, the radius of Earth and Moon is not strictly tied to mass, as density also plays a role.

The full metamodel underlying these concepts is provided in the Appendix, which offers a detailed view complementing the high-level representation in Fig. 2.

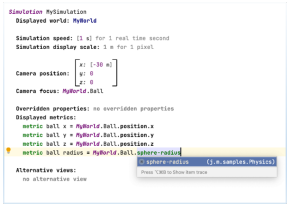

Figure 3: Creating a new Simulation concept

3.3 Language usage

To use the proposed DSL, you need to create an MPS module that utilises the necessary languages. The standard approach is to create a new solution within an MPS project and add a model inside it, ensuring that j.m.s.Physics.devkit.java is imported into the model's language dependencies (specified in the model properties).

To create your first simulation, you need to define two objects based on the following concepts:

- **World**: This object contains the entities in your simulation, along with the properties and forces applied to them, as illustrated in Figure 3.
- **Simulation**: This object defines the settings for running the simulation, such as selecting the world to display (which can reference the previously created World), as shown in Figure 4.

Once these objects are set up, you can run the Java simulation directly from the Simulation concept. The necessary code to execute the simulation will be automatically generated based on the objects and configurations specified in the model.

Please note that all user inputs are strictly related to the physics domain, with no software programming commands appearing in the interface. This approach ensures accessibility for users without programming expertise.

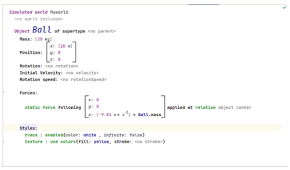

Figure 4: Creating a new World concept

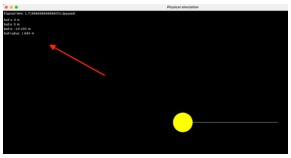

Figure 5: Simulation results

The results of the simulation are depicted in Figure 5. Additionally, in this example, metrics have been utilised to extract data from the simulation. For instance, suppose you want to place another object along your object's trajectory but are unsure where to position it. The simulation results provide this data, using appropriate units when possible (typing "Space" during the simulation will pause it, allowing for a clearer view of the data).

4 Innovation Points in Physics4All DSL

Physics4All DSL provides a domain-specific approach that makes simulation design more accessible while retaining the power to model advanced physics. This section summarises the main innovation points.

4.1 Customisable Forces and Reuse

The DSL enables users to define custom forces in terms of live object properties such as mass, position, and velocity. These forces can be reused across models, supporting clarity and consistency. Abstract objects extend this principle: a general *Planet* object may encapsulate default gravitational behaviour, while specialised versions such as *Earth* or *Moon* can override parameters. This design encourages modularity and reduces duplication in simulation models.

4.2 Multi-Target Generation and Comparison

Simulations are platform-independent and can be generated to Java or JavaScript without altering the original model. This allows the same simulation to run in both desktop and web-based environments. Educators can also generate alternative versions of a simulation to explore "what-if" scenarios, such as varying the mass of an object, and compare outcomes side by side. These capabilities make Physics4All especially suited for classroom demonstrations and interactive learning.

4.3 Generator Planning and Abstraction

Physics4All uses MPS generation plans to map high-level constructs (*Simulation*, *World*, *Force*) to executable artefacts. This separation of language and runtime improves maintainability and makes the framework easier to extend. Language engineers can focus on evolving the DSL itself, while developers can adapt runtime components for specific platforms or libraries. This modularity is essential for long-term sustainability and broader adoption.

4.4 Units, Vectors, and Live Checking

Physics4All integrates units and dimensions directly into the language. Implicit conversions within the same dimension simplify model definitions and reduce errors (e.g., converting metres to kilometres). Users may also introduce new dimensions, which are normalised to base units at runtime. For 3D simulations, the DSL supports vector and matrix operations, aligned with common coordinate systems and physics notation. Importantly, the editor provides live type checking: errors such as mismatched units or cyclic dependencies are flagged immediately, ensuring correctness before execution and improving user confidence.

5 EVALUATION

Our evaluation combines two complementary approaches. First, we conduct a **metrics-based comparison** of the implementation effort required to express a representative simulation in a general-purpose language (GPL) versus the proposed DSL (Section 5.1). This quantitative perspective highlights conciseness and clarity gains from the DSL. Second, we perform an **empirical case study** with secondary school teachers and educational technology developers (Section 5.2), using established quality characteristics to assess usability, expressiveness, and adoption challenges. Together, these evaluations provide both technical and human-centred evidence of the DSL's effectiveness.

5.1 Metrics-Based Comparison: GPL vs DSL

To demonstrate the benefits of using the proposed DSL over a general-purpose language (GPL), we present a concrete simulation example of a ball falling onto a flat surface and bouncing off it, with real-time metrics tracking the ball's velocity and energy. The source code required for both the GPL and the DSL is provided and compared.

Figure 6 shows the simulation outcome for this concrete example.

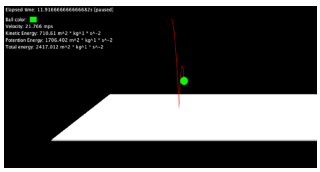

Figure 6: Concrete simulation example for comparison between DSL and GPL

To define such a simulation, we examine two scenarios: one where we use existing simulation and rendering libraries, and another where we use the proposed DSL.

5.1.1 Scenario 1: Programmatically Defining the Simulation Using GPL

When it comes to programming simulations, and more specifically visual simulations, two key components are required: the simulation engine (which computes what happens in the simulation domain) and the visualisation tool (which shows a representation of the simulation to the user). In these scenarios, we focus on a 3D view of the simulation; however, visualisation can also take the form of graphs, and the simulation data can be used for other purposes, including exporting data and computing metrics.

To create the simulation mentioned above, one must programmatically set up all the initial properties and behaviour of the simulation. Figure 7 shows a sample of the code used to set up a simulation using the simulation library, which defines all the initial parameters for the ball and the ground. This is just a short sample from a ˜250-line code file and does not include the code necessary to display the objects.

We can see that the code contains many details that may not be clear to a physical domain expert, such as the meaning of certain objects (e.g., what is a `DMass`? Why is "box" set twice at the end?) or the significance of the numbers (e.g., what physical units are being used?). In this case, the simulation creator needs to have knowledge of both a programming language and the libraries used to achieve the desired result.

```
public void setup() {
  surface.setResizable(true);
  surface.setTitle("Physical simulation");

  background( rgb: 0);
  fill( rgb: 255);
  text( str: "loading textures and setting up properties...", x: 5, y: 15);

  world = OdeHelper.createWorld();
  space = OdeHelper.createHashSpace();
  jointGroup = OdeHelper.createJointGroup();

  // Init colliders
  OxGeom.dInitColliders();

  ballBody = OdeHelper.createBody(world);
  ballBody.setPosition( x: -90, y: -5, z: 0);
  ballBody.setLinearVel( x: 2, y: 0, z: 0);
  var ballMass = 3;
  var ballRadius = 7;
  DMass mass = OdeHelper.createMass();
  mass.setMass(ballMass);
  mass.setSphere( density: ballMass / Math.PI * Math.pow(ballRadius, 3) * 4 / 3, ballRadius);
  ballBody.setMass(mass);
  // Set the body of this fixture geometry
  var geometry :DSphere = OdeHelper.createSphere(space, ballRadius);
  geometry.setBody(ballBody);


  var gndX = 500;
  var gndY = 2;
  var gndZ = 200;
  groundBody = OdeHelper.createBody(world);
  groundBody.setPosition( x: 0, y: 151, z: 0);
  groundBody.setLinearVel( x: 0, y: 0, z: 0);
  DMass gndMass = OdeHelper.createMass();
  gndMass.setMass(1);
  gndGMass.setBox( density: 1, gndX, gndY, gndZ);
  groundBody.setMass(mass);
  // Set the body of this fixture geometry
  var gndGeometry :DBox = OdeHelper.createBox(space, gndX, gndY, gndZ);
  gndGeometry.setBody(groundBody);
```

Figure 7: Code for simulation using GPL

The freedom provided by a general-purpose programming language acts as a double-edged sword: it offers complete freedom over the simulation, but it does not provide clear, direct guidance on how to build one (the code may be easy to read but difficult to write) (Tozzi, 2017).

5.1.2 Scenario 2: Defining the Simulation Using the Proposed DSL

In Figure 8, the same use case is displayed, with initial parameters defined using the DSL's mathematical notations (note that the previously displayed code corresponds to the beginning of the Ball object definition).

```
Simulated world FallingBallWorld
<no world included>

Object Ball of supertype <no parent>
  Mass: [1 kg]
  Position: [ x: -[90 m]
              y: [-5 m]
              z: 0 ]
  Rotation: <no rotation>
  Initial Velocity: [ x: [2 mps]
                      y: 0
                      z: 0 ]
  Rotation speed: <no rotationSpeed>

  Forces:
    use force Friction(ratio = 0.1) applied_at object center
    static force following [ x: 0.0
                             y: Ball.mass
                             * 9.81
                             * [1 s^-2 * m^1]
                             z: 0 ] applied at object center

  Styles:
    collision-reaction : elastic reaction(restitution: 80 %)
    texture : use colors(fill: green, stroke: <no stroke>)
    sphere-radius : [7 m]
    trace : enabled(color: red , infinite: true)

Object Ground of supertype <no parent>
  Mass: [1 kg]
  Position: [ x: 0
              y: [151 m]
              z: 0 ]
  Rotation: <no rotation>
  Initial Velocity: <no velocity>
  Rotation speed: <no rotationSpeed>

  Forces: inherited forces
  Styles:
    box-size-x : [500 m]
    box-size-y : [2 m]
    box-size-z : [200 m]
    collision-reaction : ignore
    shape : box
```

Figure 8: Code for simulation using the Physics4All DSL

In this case, no specific programming knowledge is required to write the parameters of the simulation, aside from physics and mathematics notations (e.g., vectors, mathematical operations, dimensions). This is a fundamental benefit provided by the domain focus and the reasoning behind the requirement for DSLs as

opposed to GPLs (Tozzi, 2017).

5.1.3 Comparison of the Two Scenarios

In general, using a DSL to define a simulation brings much more clarity for the user compared to using a GPL. It removes many implementation details that would otherwise need to be programmed, such as ensuring that what we see is what we simulate, and setting up the libraries (the DSL limits configuration to the necessary simulation aspects). On the other hand, the DSL may be too restrictive in some areas (missing features, structural problems), but these issues can be addressed by the language designer, much like in regular simulations that require developers to be involved.

To further illustrate this, we compared the implementation effort between the GPL and DSL solutions (Table 1). The GPL version required approximately 250 lines of code for simulation setup, while the DSL version required no user-authored GPL code, instead relying on a small number of high-level model elements. This highlights the reduction in implementation effort and the alignment of the DSL with domain concepts.

5.2 Empirical evaluation

We evaluated the Physics4All DSL based on the quality characteristics defined in the paper, as well as feedback from physics teachers and educational technology developers.

5.2.1 Study design: Objectives and Research Questions

Our research follows the case study methodology as defined by Runeson and Höst (Runeson and Höst, 2009) and aligns with the exploratory purpose of research as classified by Robson in (Robson, 2002). Specifically, it is an interpretive case study that seeks to understand phenomena through the participants' interpretation of their context. Employing a mixed methods approach as described by Robson (Robson, 2002), we collected both quantitative and qualitative data to provide a comprehensive evaluation. For the specifics of the quality criteria, we adhered to the approach outlined in Challenger's work (Challenger et al., 2016). The study is characterised as a fixed design, consistent with the classifications by Anastas (Anastas, 2000) and Robson (Robson, 2002). To enhance the robustness of our findings, we applied triangulation, involving multiple ob- servers and multiple research methods such as the integration of both quantitative and qualitative methods.

The objective of this case study is to assess the effectiveness of the developed DSL from the perspective of physics educators and educational technologists professionals. Specifically, we aim to evaluate its suitability for modeling physics simulations, ease of use for non-technical users, advantages compared to traditional documentation, and learnability. To establish a clear direction and scope for our study, we formulated four key research questions:

- **RQ1:** How does the Physics4All DSL support physics educators in creating customised experiments compared to existing simulation environments?
- **RQ2:** How does the Physics4All DSL support educational technology tool developers in creating and integrating customised experiments to existing software platforms?
- **RQ3:** What are the main usability and maintainability challenges of the Physics4All DSL, and how can they be improved for broader adoption?
- **RQ4:** How does the expressiveness and reusability of the Physics4All DSL impact its effectiveness in educational settings?

These research questions aim to comprehensively evaluate the Physics4All DSL from multiple perspectives. RQ1 examines its effectiveness in enabling physics educators to create customised experiments, assessing whether it provides the necessary flexibility and functionality compared to existing simulation environments. RQ2 explores how the DSL supports educational technology tool developers in integrating simulations into existing software platforms, evaluating its adaptability and ease of integration. RQ3 focuses on the usability and maintainability challenges of the DSL, identifying potential barriers to adoption and areas for improvement. Finally, RQ4 investigates the expressiveness and reusability of the DSL, assessing how these factors influence its effectiveness in educational settings and whether they contribute to a more efficient and scalable simulation development process.

5.2.2 Evaluation process

We evaluated the developed DSL from the perspective of non-technical physics education teachers and the education technology tool developers. For this evaluation, we followed the quality characteristics outlined in (Challenger et al., 2016), a method we have consistently applied to our DSL assessments (Meacham et al., 2020b) (Meacham et al., 2020a) (Meacham et al., 2021). However, in this study, we adapted our approach to better align with the specific requirements

Table 1: Effort comparison between GPL and DSL implementations (bouncing ball example).

Indicator	GPL (baseline)	Physics4All DSL
Implementation size	˜250 LOC	0 LOC (user code)
Model size	—	6–8 model elements
Edit surface	1–2 source files	2 roots (*World*, *Simulation*)
Pre-run defects	Compilation errors possible	Live type checks

of both the DSL and its domain-users. Our evaluation process consisted of three key elements: an instruction document provided to participants for guidance on using the DSL, the actual experiment, and a concluding questionnaire to gather final feedback.

As a first step, the DSL designer provided a brief verbal introduction along with an instruction document, which was also shared with the users as a task definition for the training.

In the second step, non-technical physics teachers (domain-users) and the educational technology developers followed the instruction document—each group receiving a tailored version. Throughout this process, the language developer took notes to assess the DSL's usability. This approach aligned with Krug's usability testing (Krug et al., 2014), though simplified to suit our requirements. Observations showed that domain-users could learn the DSL relatively quickly, despite some initial mistakes, and that the editor and its built-in hints were well received.

Finally, participants were asked to complete a questionnaire to provide feedback on the process. The questionnaire was administered electronically, with a link provided at the end of the workshop. The questionnaire included quantitative data using a Likert scale (McSkimming et al., 2021), ranging from 1 (very easy) to 5 (very difficult). To ensure feasibility, the total number of questions was limited. Additionally, open-ended questions were incorporated to allow participants to provide detailed feedback.

Examples of such questions included: "Are there any other mechanisms you would like to be supported that are currently missing?" and "Where do you think this could be used in an education practice?" These domain-specific questions aimed to gather insights relevant to the problem being addressed.

The collected responses came from a total of 10 participants, including 5 physics teachers and 5 educational technology tool developers.

5.2.3 Results - Discussion

Physics educator perspective The results from the two different user groups were very similar, and therefore, they are collectively presented as follows:

- **Functional suitability**: The language scored relatively high in terms of suitability, especially for the Physics teachers. It enabled them to create experiments that were customised to their needs.
- **Usability**: Regarding usability, the language scored medium, as the interface was not intuitive at times.
- **Reliability**: The language was very reliable, as it contained correct-by-construction elements.
- **Productivity**: The use of this language enabled a quick turnaround time for several experiments.
- **Compatibility**: This aspect was not evaluated for teachers but was scored medium by the tool developers. More openness to extensions should have been provided.
- **Expressiveness**: The language is highly expressive for the specific task.
- **Reusability**: This aspect scored surprisingly low, as the reusability of the language was difficult to understand.
- **Maintainability**: It scored low for maintainability, as it would require a background in language engineering to maintain it.

Educational technologist tool developer perspective The results from the two educational technologist tool developer perspective were as follows:

- **Functional suitability**: The language scored relatively high in terms of suitability as it contained the most commom functions required for physics simulations.
- **Usability**: Regarding usability, the language scored medium, as the interface was not intuitive at times.
- **Reliability**: The language was very reliable, as it contained strong corrections as you type mechanisms.
- **Productivity**: The use of this language enabled a quick turnaround time for several experiments. That would be particularly useful when integrating this with other education software.

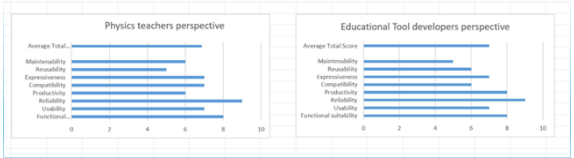

Figure 9: Evaluation by Physics teachers and education technology developers

- **Compatibility**: This aspect was rated medium by tool developers, who highlighted potential challenges in integrating the DSL with commercial tools. However, integration with other open-source, Java-based solutions would likely be more straightforward. Greater openness to extensions would have enhanced the DSL's adaptability and interoperability.
- **Expressiveness**: The language is highly expressive for the specific task.
- **Reusability**: This aspect was rated medium, with reasoning similar to that of compatibility, highlighting potential integration challenges and the need for greater extensibility.
- **Maintainability**: It scored low for maintainability, as it would require a background in language engineering to manage effectively. Even for a software developer, maintaining the DSL would necessitate specialised knowledge in language design and development, making long-term upkeep challenging.

The scorings for all individual elements are presented in Figure 9.

Addressing the Research Questions

- **RQ1:** How does the Physics4All DSL support physics educators in creating customised experiments compared to existing simulation environments?

The evaluation results indicate that the Physics4All DSL provides strong support for physics educators in designing customised experiments. The high functional suitability rating confirms that it enables teachers to create simulations that align with their specific instructional needs. Additionally, the DSL's high expressiveness ensures that educators can define precise and tailored physics simulations. The productivity score further highlights that the DSL facilitates a quick turnaround time for experiment creation. However, usability was rated medium, as some users found the interface unintuitive, suggesting that improvements in user experience and interface design could enhance accessibility for educators, particularly those with limited programming experience.

- **RQ2:** How does the Physics4All DSL support educational technology tool developers in creating and integrating customised experiments to existing software platforms?

For educational technology tool developers, the DSL was found to be functionally suitable, containing the most commonly required features for physics simulations. Its high productivity rating also indicates that it enables efficient development of new experiments, making it valuable for integration into existing educational software. However, compatibility was rated medium, as developers anticipated challenges in integrating the DSL with commercial software. While integration with open-source, Java-based solutions was expected to be more straightforward, the DSL would benefit from greater openness to extensions to enhance adaptability. Reusability also scored medium, reflecting the need for better modularity and documentation to facilitate seamless integration with external platforms.

- **RQ3:** What are the main usability and maintainability challenges of the Physics4All DSL, and how can they be improved for broader adoption?

Both user groups rated usability as medium, indicating that while the DSL is functional, its interface is not always intuitive. This presents a potential barrier to adoption, particularly for educators unfamiliar with programming environments. Maintainability scored low, primarily due to the requirement for a background in language engineering to effectively manage and extend the DSL. Even for experienced software developers, maintaining the DSL necessitates specialised knowledge in language design and development, increasing its complexity. To broaden adoption, the DSL should focus on enhancing usability through a more intuitive interface and reducing maintenance overhead by simplifying the underlying architecture and providing better documentation.

- **RQ4:** How does the expressiveness and reusability of the Physics4All DSL impact its effectiveness in educational settings?

The high expressiveness rating demonstrates that the DSL effectively supports custom physics simulations, allowing educators and developers to create experiments with advanced customisation. However, reusability scored low among educators and medium among developers, indicating difficulties in repurposing components across different simulations. This suggests that improvements in modular design and

comprehensive documentation are needed to facilitate the reuse of existing components, reducing redundancy and making the DSL more scalable for broader applications.

6 COMPARISON WITH EXISTING SIMULATION ENVIRONMENTS

We contrast Physics4All with PhET (PhET Interactive Simulations, 2025), Interactive Physics (Design Simulation Technologies, 2025), Algodoo (Algodoo, 2023), Easy Java Simulations (de Murcia, 2025), and COMSOL Multiphysics (Inc., 2025), using three criteria: (i) customisability/support for advanced physics; (ii) required programming knowledge; (iii) cost.

Summary. Physics4All offers high customisation *without* programming, plus open multi-target generation for classroom deployment and platform integration. It complements PhET for introductory exploration by enabling educator-authored, curriculum-aligned experiments.

7 Limitations of the Current Work

While the results of this study demonstrate the potential of Physics4All, several limitations should be acknowledged.

First, the empirical evaluation involved a small sample of participants (ten in total: five teachers and five educational technology developers). Although this provided valuable insights into usability and suitability, the findings cannot be generalised to the wider community of physics educators without further large-scale studies. Future work should aim to replicate the study with more diverse participant groups across different educational contexts.

Second, the current implementation of Physics4All is an early-stage prototype. While it already supports key features such as units, dimensions, and vector operations, additional engineering effort is needed to improve maintainability, documentation, and the user interface. This was also reflected in participant feedback, where maintainability and usability were rated lower than other criteria.

Third, the evaluation focused on functional suitability and usability but did not investigate long-term adoption or integration into existing educational platforms. Compatibility with commercial software remains an open issue, particularly for tool vendors who may require broader extensibility and interoperability than currently offered.

Finally, the present work concentrated on physics simulations for secondary education. While the concepts could be extended to other domains or to more advanced areas of physics, these applications were not within the scope of the current study.

8 CONCLUSIONS AND FUTURE WORK

In this paper, we presented Physics4All, a domain-specific language (DSL) for designing and developing physics simulations tailored for secondary school teachers and educational technology developers. The DSL was created to democratise simulation creation by enabling domain-users to develop simulations without requiring extensive programming knowledge. While various physics simulation tools exist, none effectively combine customisability, ease of use, and advanced physics modelling for educators and developers.

Our evaluation combined a **metrics-based comparison** of GPL and DSL implementations with an **empirical case study** involving teachers and educational technology developers. The metrics-based study demonstrated significant reductions in implementation effort and clearer alignment with physics concepts, while the case study confirmed high functional suitability, expressiveness, and productivity. Both strands also revealed limitations: usability challenges in the interface, compatibility and reusability issues for integration, and maintainability concerns due to the need for language engineering expertise.

Compared to existing simulation environments such as PhET, Algodoo, and COMSOL Multiphysics, Physics4All stands out for its customisability and support for advanced physics features, while remaining more accessible than code-based alternatives. However, improvements in reusability and modularity are necessary to ensure seamless integration with commercial and open-source educational tools.

Beyond physics education, the innovations in Physics4All—such as modular generators, reusable domain abstractions, and integrated unit checking—illustrate principles that can generalise to other DSLs. These engineering contributions highlight how domain-specific languages can balance accessibility for non-programmers with rigorous model-driven engineering practices.

Future work will focus on larger-scale studies to refine the DSL's usability and integration capabilities. Planned enhancements include modular design, im-

proved documentation, and a more intuitive interface, all aimed at making the DSL more accessible to a broader audience. Furthermore, we intend to explore its applicability beyond physics education, extending its impact to other simulation-driven domains. By addressing these challenges, Physics4All has the potential to become a widely adopted tool for both educators and technology developers.

REFERENCES

Abar, S., Theodoropoulos, G. K., Lemarinier, P., and O'Hare, G. M. (2017). Agent based modelling and simulation tools: A review of the state-of-art software. *Computer Science Review*, 24:13–33.

Algodoo (2023). Algodoo - Physics Simulation Software. Accessed: May 08, 2023.

Anastas, J. W. (2000). *Research design for social work and the human services*. Columbia University Press.

Banda, H. J. and Nzabahimana, J. (2023). The impact of physics education technology (phet) interactive simulation-based learning on motivation and academic achievement among malawian physics students. *Journal of Science Education and Technology*, 32(1):127–141.

Ben Ouahi, M., Lamri, D., Hassouni, T., Ibrahmi, A., and Mehdi, E. (2022). Science teachers' views on the use and effectiveness of interactive simulations in science teaching and learning. *International Journal of Instruction*, 15(1):277–292.

Campos, N., Nogal, M., Caliz, C., and Juan, A. A. (2020). Simulation-based education involving online and on-campus models in different european universities. *International journal of educational technology in higher education*, 17:1–15.

Çelik, B. (2022). The effects of computer simulations on students' science process skills: Literature review. *Canadian Journal of Educational and Social Studies*, 2(1):16–28.

Challenger, M., Kardas, G., and Tekinerdogan, B. (2016). A systematic approach to evaluating domain-specific modeling language environments for multi-agent systems. *Software Quality Journal*, 24:755–795.

de Murcia, U. (2025). Easy java simulations (ejs). Accessed: 24 February 2025.

de Paula Ferreira, W., Armellini, F., and De Santa-Eulalia, L. A. (2020). Simulation in industry 4.0: A state-of-the-art review. *Computers & Industrial Engineering*, 149:106868.

Design Simulation Technologies (2025). Interactive physics: Physics simulation software for the classroom. Accessed: February 24, 2025.

Inc., C. (2025). Comsol multiphysics® modeling software. Accessed: 24 February 2025.

Krug, S. et al. (2014). Don't make me think, revisited. *A Common Sense Approach to Web and Mobile Usability*.

McLeish, D. L. (2011). *Monte Carlo simulation and finance*, volume 276. John Wiley & Sons.

McSkimming, B. M., Mackay, S., and Decker, A. (2021). Investigating the usage of likert-style items within computer science education research instruments. In *2021 IEEE Frontiers in Education Conference (FIE)*, pages 1–8. IEEE.

Meacham, S., Pech, V., and Nauck, D. (2020a). Adaptivevle: An integrated framework for personalized online education using mps jetbrains domain-specific modeling environment. *IEEE Access*, 8:184621–184632.

Meacham, S., Pech, V., and Nauck, D. (2020b). Classification algorithms framework (caf) to enable intelligent systems using jetbrains mps domain-specific languages environment. *IEEE access*, 8:14832–14840.

Meacham, S., Pech, V., and Nauck, D. (2021). Adaptivesystems: An integrated framework for adaptive systems design and development using mps jetbrains domain-specific modeling environment. *IEEE Access*, 9:127973–127984.

Mernik, M., Heering, J., and Sloane, A. M. (2005). When and how to develop domain-specific languages. *ACM computing surveys (CSUR)*, 37(4):316–344.

Ogegbo, A. A. and Ramnarain, U. (2022). Teaching and learning physics using interactive simulation: A guided inquiry practice. *South African Journal of Education*, 42(1).

PhET Interactive Simulations (2023). PhET: Free online physics, chemistry, biology, earth science and math simulations. Accessed: May 08, 2023.

PhET Interactive Simulations (2025). Phet: Free online physics, chemistry, biology, earth science, and math simulations. Accessed: February 24, 2025.

Rehman, N., Zhang, W., Mahmood, A., and Alam, F. (2021). Teaching physics with interactive computer simulation at secondary level. *Cadernos de Educação, Tecnologia e Sociedade*, 14(1):127–141.

Robson, C. (2002). *Real World Research*. Blackwell.

Runeson, P. and Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering*, 14:131–164.

Tozzi, C. (2017). *For fun and profit: A history of the free and open source software revolution*. MIT Press.

Vaclav (2025). Physics github repository. Accessed: 26 September 2025.

Voelter, M. (2017). High-level structure of dsls: Three patterns. Accessed: Feb 4, 2025.

Völter, M. (2018). The design, evolution, and use of kernelf: An extensible and embeddable functional language. In *Theory and Practice of Model Transformation: 11th International Conference, ICMT 2018, Held as Part of STAF 2018, Toulouse, France, June 25–26, 2018, Proceedings 11*, pages 3–55. Springer.

Völter, M. (2020). Kernelf – an embeddable and extensible functional language. https://voelter.de/data/pub/kernelf-reference.pdf. Accessed: 24 February 2025.

APPENDIX: Physics4All DSL metamodel