Extending ADL/CutLang with a new dynamic protocol

Gokhan Unel[1], **Grigory Fedyukovich**[2], **Berare Gokturk**[3], **Harrison B. Prosper**[2], **Daniel Riley**[2], **Sezen Sekmen**[4], **Burak Sen**[5]

[1] UC Irvine
[3] Florida State University
[2] Bogazici University
[4] Kyungpook National University
[5] Middle East Technical University

E-mail: `gokhan.unel@cern.ch`

Abstract.
We present the latest developments in Analysis Description Language (ADL), a declarative domain-specific language describing the physics logic of a particle physics data analysis independent of frameworks, and CutLang, its runtime interpreter. Alongside continuous ADL syntax refinements through implementation of real world analysis examples, static program analysis tools for ADL are being developed, starting with a standalone graphviz-based tool to generate analysis logic graphs. For CutLang, a major revision to enable decoupling input data information and external functions from the core ADL syntax is underway. We also introduce the newly developing Dynamic Domain Specific eXtensible Language (DDSXL) protocol that generalizes the use of ADL and CutLang to multiple domains. DDSXL hosts numerous programming languages and frameworks. Domain ecosystems can be integrated into abstract DDSXL development environment using various OOP design patterns and a set of rules determined through communication over the network.

1. Introduction

The CERN Large Hadron Collider (LHC) experiments already published several thousand physics analyses, reflecting a diverse and creative physics program. These analyses have broadly similar workflows that consist of dataset handling, event processing, visulization and statistical analysis. The physics logic in these analyses includes defining analysis objects, defining quantities based on event properties, selecting events, re-weighting simulated events to improve their agreement with real collision events, estimating backgrounds and interpreting experimental results by comparing them to predictions. Workflow management and physics logic are traditionally implemented in software frameworks coded based on general purpose languages (GPLs) such as C++ / Python. Yet, the framework architectures largely vary from experiment to experiment, even analysis to analysis. This technical diversity fosters inventiveness, but it may also hinder communication and long term preservation of analysis details.

In recent years, using declarative or domain-specific languages (DSLs) have emerged as an alternative approach to imperative programming with GPLs for a clearer and more systematic expression of analysis workflows or physics logic. DSLs are inherently self-documenting and decouple from the backend implementation, allowing updates to the backend as new technologies become available. Declarative programming, on the other hand, by definition allows direct expression of the logic via a higher-level programming, without an explicit description of the control flow. These features can be particularly advantageous in physics logic description as they enable presenting the physics information in a way decoupled from execution details. Consequently, declarative or domain-specific programming approach is being increasingly adopted in developing infrastructures for physics analysis.

In this note, we will focus on one prominent example, Analysis Description Language (ADL) and its runtime interpreter infrastructure CutLang, which provide a multipurpose setup for a wide range of studies including analysis design, reinterpretation, preservation, visualization, comparison, etc. After introducing the language and the interpreter, we will discuss the latest technical developments. Then, we will present Dynamic Domain Specific eXtensible Language (DDSXL), a new infrastructure to generalize the use of ADL and CutLang to multiple domains.

2. Analysis Decription Language

Analysis Description Language (ADL) is a declarative domain specific language (DSL) that describes the physics content of a HEP analysis in a standard and unambiguous way [1, 2, 3]. It is an external DSL, with custom-designed syntax to express analysis-specific concepts, reflecting conceptual reasoning of particle physicists. ADL is designed as a generic, multipurpose construct for users with different goals and levels of expertise. It can be used by experimentalists for full scale data analysis, phenomenologists for reinterpretation or sensitivity studies, students for education, or the wider public for familiarizing with HEP or studies on open data.

ADL is designed to be framework-independent, meaning that any framework recognizing the ADL syntax can perform tasks with it. This independence was adopted to decouple physics information from software / framework details and present the physics logic in a standalone way. This enables a multipurpose use of ADL, as it can be automatically translated or incorporated into the GPL or framework most suitable for a given purpose. It also allows easy communication between different groups, such as different analysis groups, experiments and phenomenologists, etc., and facilitates analysis logic preservation. Drawing an analogy between analysis code development and building construction, working directly in a GPL would be equivalent to starting from the foundation below the ground. If we opt to use an established framework from an experiment it might be analogous to starting from an existing building where there is already a fountation and few floors but there is also the old plumbimg and wiring that comes with it. In contrast, using ADL is similar to modular construction, i.e. creating a building with a sturdy foundation, constantly updated infrastructure and pre-made ready to install floors.

ADL scope is centered on describing the physics logic in event processing. It includes definitions of simple and composite objects (jets, muons, top quarks, etc.), event variables (transverse mass, effective mass, etc.), optimizations (χ^2 optimization for reconstructing the best top quark pair), event selections ($n_{electrons} > 0$, $m_T > 140$, etc.) and event weighting. Expression of systematic uncertainties would also be included within the ADL scope, however the syntax for these is still under development. Histograms and expressions of existing analysis results such as counts and uncertainties, are also a part of the ADL scope. However details of background estimation procedures and statistical analysis are outside the scope.

The analysis description written in ADL syntax is contained within a plain, easy-to-read text file called the ADL file. This file consists of multiple types of blocks separating different analysis components such as object, variable and event selection definitions. Blocks adopt a keyword-expression structure, where keywords specify analysis concepts and operations. Apart from the DSL keywords, the current syntax includes mathematical and logical operations, comparison and optimization operators, reducers, 4-vector algebra and standard HEP functions (e.g. $\Delta\phi$, ΔR). One and two-dimensional histograms with fixed or variable bins for object and event quantities can also be defined. Externally available analysis output including event counts and uncertainties (e.g. published by experiments) can also be documented in the ADL file. Some analyses may contain variables with complex algorithms non-trivial to express with the ADL syntax (e.g. stransverse mass, aplanarity) or non-analytical, heavily numeric functions (e.g. machine learning models, efficiency tables). Such variables are encapsulated in self-contained, standalone external functions written in a GPL that can be referenced from within an ADL file. Further details and example implementations of the ADL syntax are documented in [10].

ADL's ongoing developement has continuously been accompanied by the transcription of ATLAS and CMS analyses from a wide spectrum of physics areas (e.g., Higgs, Top, SUSY, exotics). A growing repository of ADL analysis transcriptions can be found in [4]. This allows to test the capacity of existing syntax to describe state-of-the art analysis operations, while determining and adressing further syntactic requirements. A recent study through an ongoing ATLAS heavy lepton analysis lead to a prototype implementation of systemeatic uncertainties with an ATLAS-style treatment, where the up and down variations are typically stored in ntuples, and implemented as event weights. Work continues towards syntax generalization.

2.1. Analysis visualization with ADL

One significant virtue of adopting a well-defined declarative DSL for HEP analyses is the potential of performing static program analysis, i.e. analysis of source code without its execution, for a variety of purposes. One implication is to obtain a visualisation of the analysis physics logic via graphs automatically generated from the ADL file. A prototype auto-graph generator was recently implemented based on the `graphviz` package, and is available through Visual Studio Code [5]. ADL files undergo simple parsing, and converted a a description in the native `dot` language of `graphviz`. The graph represents all input objects, derived objects and regions as nodes, and shows their connections to each other. Figure 1 shows the graph representation of a CMS SUSY analysis [6] obtained by the above mechanism. Work is ongoing to develop this prototype into a more mature tool based on a formal ADL syntax parsing.

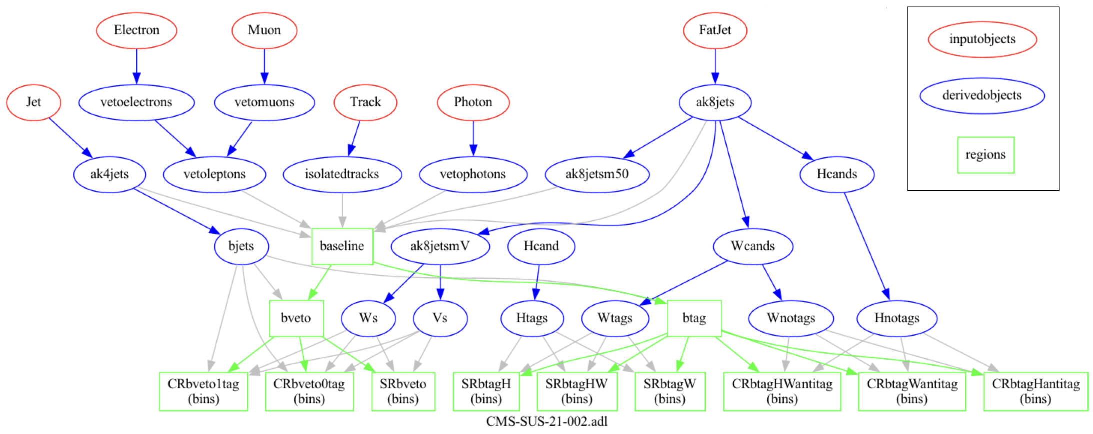

Figure 1. Auto-generated graph representation of the CMS search for electroweak SUSY in final states with hadronic, boosted WW, WZ and WH [6] from the ADL file [7] via a graphviz-based tool prototype [5].

ADL helps to design and document a single analysis in a clear and organized way, but its distinguishing strength is in navigating and exploring the multi-analysis landscape, for which static analysis provides an essential tool. For example, static analysis can assist and automate query among, or comparison between multiple analyses in the space of event properties. It can be used to determine analysis overlaps, identify disjoint analyses or search regions, which in turn can help to find the feasible combinations with maximal sensitivity and automate large scale combinations of analyses. Work towards developing such infrastructures is ongoing.

3. CutLang

CutLang is an infrastructure consisting of a run-time interpreter for ADL and auxiliary tools. It started as an exploratory protype and has matured over the years [8, 9, 10, 11]. CutLang is written in C++ and is based on ROOT classes for Lorentz vector operations and histogramming. ADL parsing is performed by the state-of-the art tools Lex and Yacc. CutLang has a large inventory of predefined and debugged functions which guarantee correct cut efficiency calculation, correct combinatorics without double counting, sorting, unions of multiple object types, etc. It also has the ability to execute algorithms with multiple regions in parallel, to cache the results of the preselection and use it in regions dependent of the preselection, to solve χ^2 optimization problems for heavy particle reconstruction from daughters, to save events at any stage of the analysis in flat ROOT ntuples or as comma separated values in text files.

Events are input via ROOT files in multiple types of formats (including Delphes, CMS NanoAOD, ATLAS/CMS Open Data, LVL0, FCC), and more can be easily added. All event types are internally converted into predefined particle object types, which allows an option to run the same ADL file on different input types. Analysis output is given in ROOT files that contain cutflows, bins and histograms for each region in a separate directory, along with the ADL file itself for provenance tracking. CutLang is available in multiple platforms, i.e. Unix, Docker, Conda, Jupyter (via Conda or binder), that allow usage in Linux, macOS and Windows, and even on partable phones and tablets. In particular, a Docker container is made available for analysis of CMS Open Data, which combines CutLang, `xrootd` and VNC. Like ADL, CutLang is continuously and extensively stress-tested by running ADL implementations of realistic analysis cases, including experimental data analysis, reimplementation of analyses for reinterpretation [12] and future collider sensitivity studies [13]. It has also been extensively used for training the next generation of students [14]. More recently, a complete CMS vector-like quark analysis based on 2015 has been implemented and processed over CMS Open Data, and a tutorial was made available within the context of CMS Open Data Workshop 2022 [15].

However, after years of coding by multiple developers, CutLang has become a large and monolitic program with its own maintenance and debugging difficulties. A redesign and reimplementation effort has recently been initiated to modernize CutLang. The most immediate requirements that have been identified for the rejuvenation effort are the following:

- Currently input data content (e.g. objects, attributes (as muon identification, b-tagging discriminant), triggers, etc.) used in event selection needs to be manually hard-coded into the CutLang syntax parser. Input data attributes must be separated from the ADL core syntax to achieve a dynamic addressing of various data formats.
- Currently external functions for calculating complex or non-analytical variables are integrated manually into CutLang. Their integration needs to be automated, and names should be separated from the core ADL syntax.

The new system currently under development will allow the decoupling of the core grammar and eliminate the need to hard-code data attributes or function names. After initial parsing, the system will match data attributes and function names to those within an external library. This will allow for a more flexible and portable system, which does not require direct maintenance on the core code, and easily link to different function implementations. Deployment of the independently developed system into the CutLang infrastructure is expected by mid-2023.

4. Dynamic Domain Specific eXtensible Language

One aspect worthwhile exploring is the applicability of ADL and CutLang principles to other domains such as non-collider or dark matter experiments. This could enable more informed global fits through a more coherent analysis of data from diverse sources. This notion led to the creation of the so called DDSXL, the Dynamic Domain Specific eXtensible Language protocol.

The main idea behind this protocol is to separate different functionalities inherited from CutLang into different services which make themselves available to a central core application. All other tools and complexities being hidden from the user, the user's only interaction happens with this DDSXL core. By seperating the DSL and its parsing from the execution engine doing the run-time interpretation, it becomes possible to address the third requirement.

Following this logic, the new tool can execute not only the algorithms of ADL but any other DSL description in any other domain. The input text based on a domain-specific syntax, prepared by the user and executed by the DDSXL core, will henceforth be called an xDL file. Each such DSL can be integrated into the DDSXL system by matching the following availability requirements: 1) a parsing service, 2) an execution engine, 3) a set of (N) libraries defining the keywords and functions of that language.

The word dynamic in the new protocol's name can be justified by making each DSL available at run-time. Similarly each language can be extended (hence eXtensible) at run time by adding another library providing new functionality. The DDSXL protocol is depicted in Figure 2. Different DSLs and the associated tools are represented by horizontal rectangles of different colors. In this notation, M such languages can be defined through M parsing and executing engines and N×M libraries.

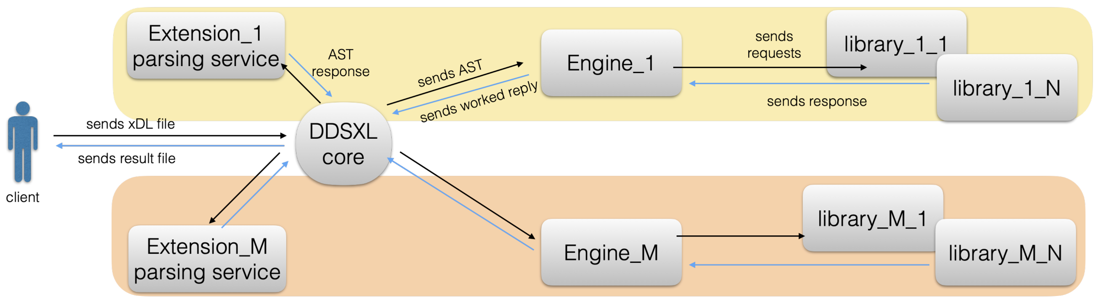

Figure 2. Schematic presentation of the DDSXL protocol.

Once the core service is running, a DSL registers itself to the DDSXL core by providing a parsing service. The goal of this service is to receive the relevant xDL file and to produce an abstract syntax tree (AST) that can be executed by that DSL's interpretation engine running as another service. The execution engine needs the definitions of the necessary functions (e.g. ΔR) of the associated DSL. The associated libraries provide exactly that, by sending responses to the definition requests. Naturally each such library needs to register itself to the execution engine as a run-time service.

By converting the whole communication protocol into a set of services around a central core, the whole system can be generalized into a network based setup. Each DSL can run on a different server or each component such an engine or a library can run on a different server or all components can run on a single machine: DDSXL is flexible enough to allow various topologies.

As of this note's writing, a prototype has been put together and basic functionality has been tested. For this prototype, the execution protocol steps and technologies to be used are identified, and include gRPC (https://grpc.io/) and GraphQL (https://graphql.org/) . Test servers and clients are also written to validate the intended functionality. The run-time library addition was successful and the DDSXL core application correctly received the keywords of the language as published by the parser. Overall, DDSXL working logic is akin to that of "terraform", which is often used in cloud computing (only more generally orchestrating for everything). In principle, the ADL/CutLang/DDSXL trio is a candidate to form an Analytics as a Service (AaaS) platform, and through interest, usage and support from the community, in long term, it could evolve into one. The development of the DDSXL protocol is ongoing.

5. Conclusions and Outlook

Declarative or domain specific languages are emerging as means to express analysis workflows or physics logic in a more systematic and clear way. Analysis Description Language (ADL) is a prominent example that demonstrates the feasibility and advantages of this approach. ADL, as a framework-independent construct enables multipurpose use and preservation of the physics logic. ADL syntax, already capable of expressing a large variety of analysis logic, is continuously being tested by real world examples, towards refinements and improvements. ADL also opens a wealth of possibilities through static program analysis in areas of analysis visualization, queries, comparisons and overlap studies. A graphviz-based prototype tool drawing analysis logic flows as graphs was presented as a first demonstration in this direction, and more are under development.

CutLang, the run-time interpreter for ADL already has the capacity to parse and process analyses of moderate complexity over multiple event formats in multiple platforms. A fundamental modernization in CutLang's core, to decouple input data type or external function names from the ADL syntax, towards enabling the capability to automatically integrate new input types and new external functions is well underway. We anticipate the use of DSLs to expand beyond the HEP community and into other fields of science, where similar analysis workflows are used. Therefore, a parallel effort is ongoing to design Dynamic Domain Specific eXtensible Language (DDSXL), a new infrastructure to generalize the use of ADL and CutLang to multiple domains. The development and adoption of the ADL/CutLang/DDSXL system have the potential to transform the way we conduct and communicate physics analysis, enabling a more systematic and transparent approach to science, and hence will be continuously explored.

Acknowledgments

The work of SS is supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education under contracts NRF-2021R1I1A3048138, NRF-2018R1A6A1A06024970 and NRF-2008-00460.

References

[1] G. Brooijmans, *et. al.* "Les Houches 2015: Physics at TeV colliders - new physics working group report," [arXiv:1605.02684 [hep-ph]].
[2] G. Brooijmans, *et. al.* "Les Houches 2017: Physics at TeV Colliders New Physics Working Group Report," [arXiv:1803.10379 [hep-ph]].
[3] H. B. Prosper, S. Sekmen and G. Unel, "Analysis Description Language: A DSL for HEP Analysis,", Contribution to: 2022 Snowmass Summer Study, [arXiv:2203.09886 [hep-ph]].
[4] ADL LHC Analyses Database, https://github.com/ADL4HEP/ADLLHCanalyses
[5] Graph generator: https://marketplace.visualstudio.com/items?itemName=shenburak.adl-cutlang-initial
[6] A. Tumasyan *et al.* [CMS Collaboration], Phys. Lett. B **842** (2023), 137460 doi:10.1016/j.physletb.2022.137460 [arXiv:2205.09597 [hep-ex]], SUS-SUS-21-002.
[7] CMS-SUS-21-002 ADL: https://github.com/ADL4HEP/ADLAnalysisDrafts/tree/main/CMS-SUS-21-002
[8] S. Sekmen and G. Ünel, "CutLang: A Particle Physics Analysis Description Language and Runtime Interpreter," Comput. Phys. Commun. **233** (2018), 215-236 [arXiv:1801.05727 [hep-ph]].
[9] G. Unel, S. Sekmen and A. M. Toon, "CutLang: a cut-based HEP analysis description language and runtime interpreter," J. Phys. Conf. Ser. **1525** (2020) no.1, 012025 [arXiv:1909.10621 [hep-ph]].
[10] G. Unel, S. Sekmen, A. M. Toon, B. Gokturk, B. Orgen, A. Paul, N. Ravel and J. Setpal, "CutLang V2: towards a unified Analysis Description Language," [arXiv:2101.09031 [hep-ph]].
[11] CutLang github repository, https://github.com/unelg/CutLang
[12] G. Brooijmans, *et. al.* "Les Houches 2019 Physics at TeV Colliders: New Physics Working Group Report," [arXiv:2002.12220 [hep-ph]].
[13] A. Paul, S. Sekmen and G. Unel, "Down type iso-singlet quarks at the HL-LHC and FCC-hh," Eur. Phys. J. C **81** (2021) no.3, 214 [arXiv:2006.10149 [hep-ph]].
[14] A. Adıgüzel, *et. al.*, "CutLang as an Analysis Description Language for Introducing Students to Analyses in Particle Physics," Eur. J. Phys. **42** (2021), 035802 [arXiv:2008.12034 [hep-ph]].
[15] Tutorial for implementation of a CMS search for vector-like quarks with ADL/CutLang for CMS Open Data, https://cms-opendata-workshop.github.io/workshop2022-lesson-run2-adlcl/