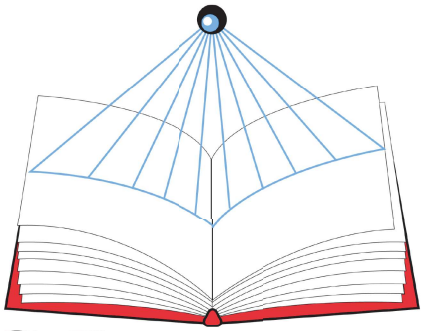


R.C. DEVELOPMENT

System Design Document

CLIF Parser, designed for Torsten Hahmann and Jake Emerson

Designed by Reading Club Development:
Matthew Brown, Gunnar Eastman, Jesiah Harris, Shea Keegan, Eli Story

Version 1.0
Nov. 9, 2022

Table of Contents

1. Introduction

In this section of this System Design Document (SDD), the project will be introduced along with the purpose of the SDD, the references used and compiled by Reading Club Development (RCD), the purpose of the product, and the scope of the product.

This is a capstone project for Dr. Torsten Hahmann and Jake Emerson, in partial fulfillment of the Computer Science BS degree for the University of Maine. This SRS will detail the functional and nonfunctional requirements set forth for this project, the deliverables necessary for completion, and document the consent of the team members and the client.

The field of biology is currently facing a “crisis of reproducibility” according to Jake Emerson, one of our clients and an engineer at Jackson Laboratory in Bar Harbor, Maine. The development of an ontological reader is, for him, paramount due to the congruity of semantics within ontological statements. The CLIF Parser is to be a stepping stone in the progress toward unified Common Logic, allowing for more properly defined variables, and hopefully slightly mitigating this crisis of reproducibility.

On a smaller scale, the purpose of the CLIF Parser is primarily to parse CLIF files to ensure they are syntactically correct according to CLIF standards. The library should also translate from the CLIF syntax to TPTP, or other logic-based conventions. Another purpose of the product is to build upon the previously developed Macleod IDE, so Common Logic can have an IDE dedicated to supporting it.

1.1 Purpose of This Document

This SDD is meant to expand upon the design details for the Common Logic Interchange Format (CLIF) Parser that RCD has been tasked with developing. The intended readership of this document consists of the client and the RCD team so as to effectively develop the proposed CLIF Parser.

1.2 References

Macleod, Dr. Torsten Hahmann, GitHub, 2022, https://github.com/thahmann/macleod.

2. System Architecture

Within this section are several design diagrams meant to illustrate in further detail the inner workings of the system. Many of the diagrams may outline the Macleod system as it currently exists, though this is to ensure that RCD does not deviate far from the current implementation of the system. Differences between the current Macleod implementation and the design diagrams highlight improvements that will be made by RCD, at the behest of the client, and in line with previously outlined functional requirements.

2.1 Architectural Design

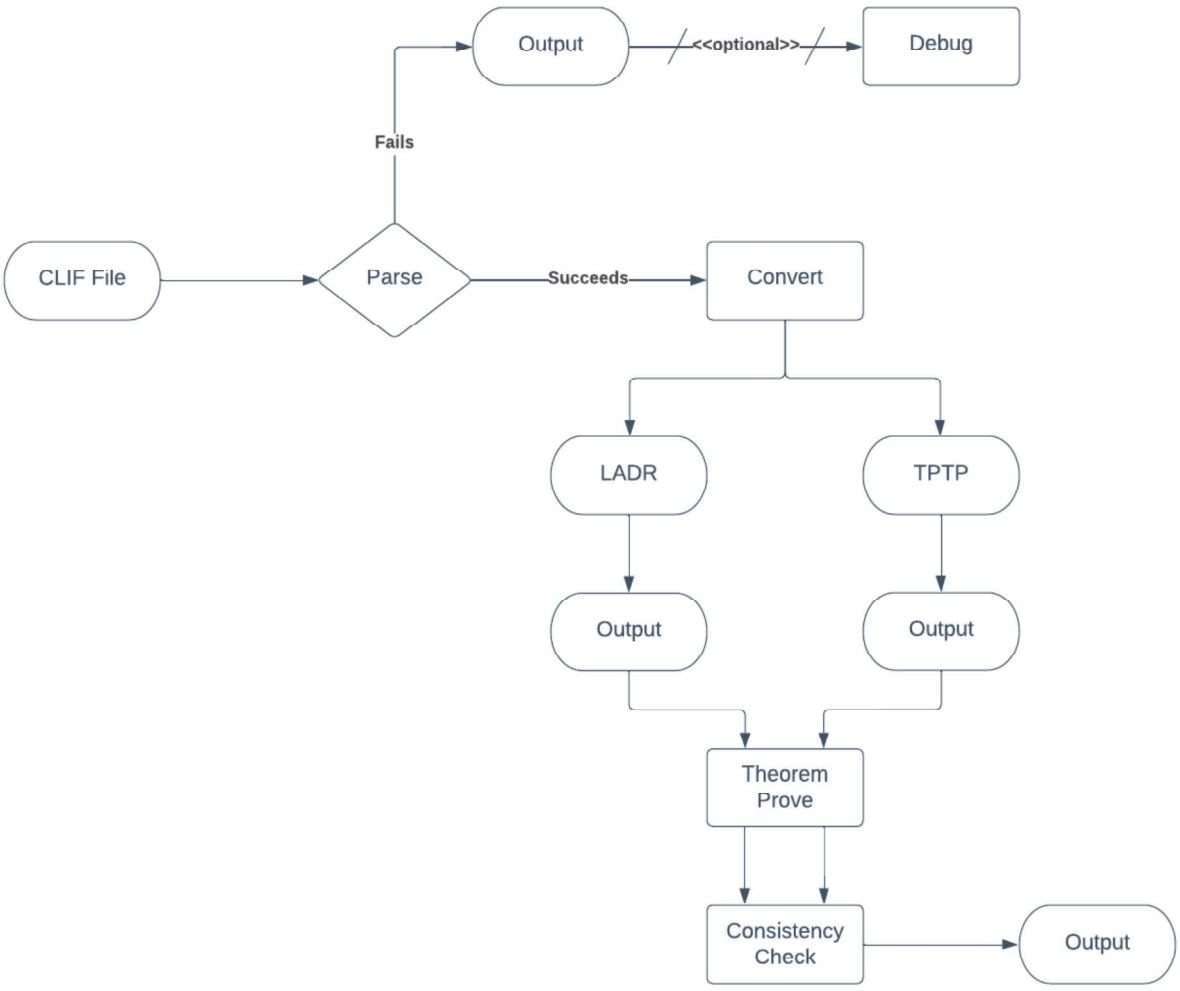


Fig. 1: Data Architecture Design of Macleod: This diagram documents the flow of data through the Macleod system, from the input of CLIF files to eventual output.

Fig. 1 is read left to right where the initial step of the system is a CLIF file being input into the Macleod system. From there, the file is parsed for errors syntactically. If the file fails

parsing, then an output occurs wherein the reason for parse failure is given, as is the option to debug the file. If the file passes the parsing, it is then converted into TPTP, LADR, and possibly OWL. Then these translations are passed through a consistency checker, and the Macleod system outputs the consistency (True or False). The CLIF files, the TPTP and LADR translations, and both outputs are visible to the user, hence their oval bubbles. The parsing, conversion, theorem provers, consistency check, and debugging are scripts or systems, thus they are rectangles.

The system will use python primarily, as the parser is written in python. The parser is built for the Macleod system, which contains the conversion scripts, theorem provers, and consistency checker. The theorem provers used will be Vampire, Prover9, and Mace4. The IDE itself will be built on Tkinter, written in Python. Macleod should function for Python versions 3.7 and newer. As for hardware, macleod should work on Windows 10 and later, Mac Monterey 12.0 and later, and Ubuntu Linux. RCD's work on Macleod will follow the Pipe and Filter architecture model, wherein data is passed through a series of steps to an eventual output, as documented in the above diagram.

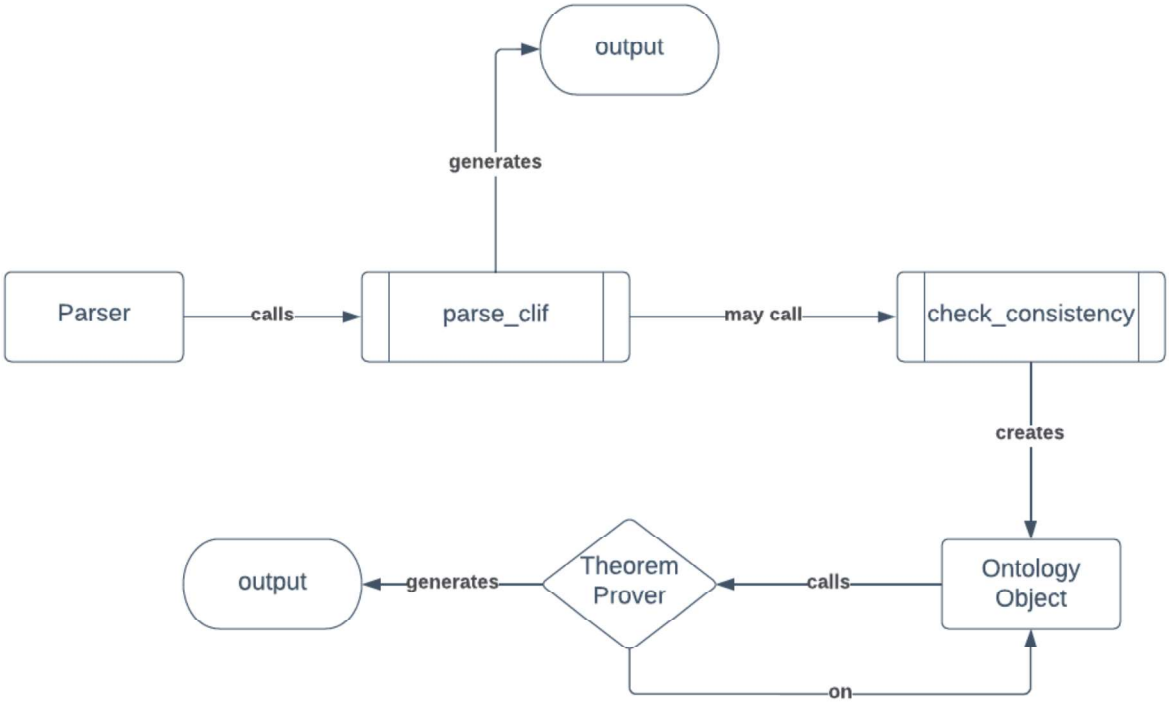


Fig. 2: Technology Architecture Diagram of Macleod: This diagram documents how scripts, theorem provers, and outputs are handled by Macleod.

Fig. 2 is read as a flowchart where the Parser and Ontology Object are objects within the system, parse_clif and check_consistnecy are scripts within the system, the outputs are user accessible, and the theorem prover checks the logical consistency of the ontology object. The purpose of this diagram is to document the various technologies at play within the entire Macleod system at a more granular level than the flow of data. The only place for

user input in this diagram is whether the check_consistency script is called.

2.2 Decomposition Description

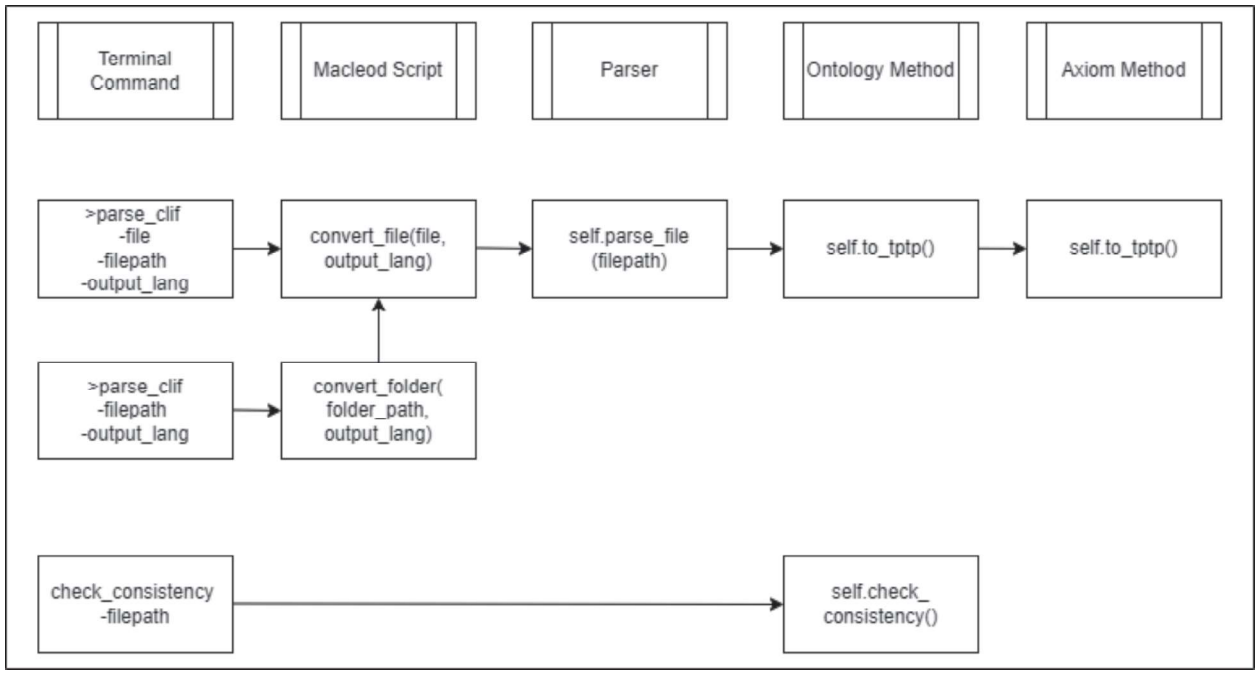


Fig. 3: Outline of Macleod function call order: This diagram describes which functions call which other functions in the parsing and translating processes.

Below are descriptions of the scripts the system makes available to users, the parameters of each of these scripts, and the functions called by each of them.

Scripts:

- parse_clif [file or directory] [filepath] [format] [optional]
    - parse_clif will call either convert_file or convert_folder, which iterates through the files and calls convert_file on each of them.
    - convert_file will create an Ontology object for the file if one does not already exist, and call the parser to read the CLIF file.
    - Once the CLIF file has been read and exists in terms of formula, axiom, and ontology objects, the convert_file function will simply call the proper translation function, according to which format to put the output in, which will be a method of the Ontology object.
    - The translation functions simply iterate through each axiom and have each one translate themselves, adding them to an output at the end.
- check_consistency [filepath] [optional]
    - check_consistency will assemble an ontology object from the input, and then call the check_consistency() method of that object.
    - The check_consistency method simply allows the ontology object to call a theorem prover on itself.

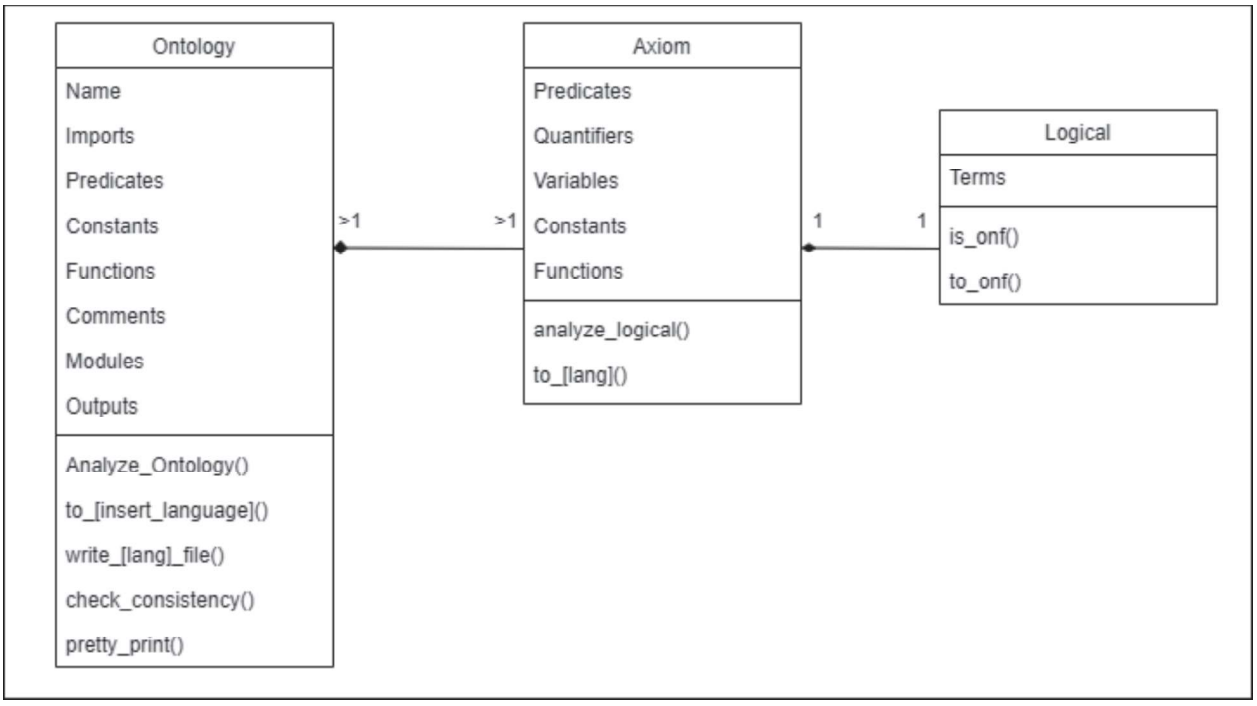


Fig. 4: Class Diagram of Macleod Objects: Describes the fields and methods of the Ontology, Axiom, and Logical Formula classes.

Three main classes are used in the system: Ontologies, Axioms, and Logical statements.

Each formula object contains an array of logical symbols, each represented by an object of a different class, though each of those objects contains very little other than a name. Each formula object is passed in as a field of an axiom object.

The axiom class will make use of its fields to keep far more detailed information about the formulas and what is in it than the formula object itself, and will be able to translate itself into different logical file formats.

Each Ontology object contains any number of these axiom objects, and will maintain a list of predicates, constants, etc. contained within its axioms for convenience, as well as being able to translate itself into different logical file formats, keep those translations should they be needed again, call theorem provers upon itself, and print itself out in the terminal for readability.

3. Persistent Data Design

This section is to display the data that our system will use to run or produce.

3.1 Database Descriptions

Databases are not applicable to the CLIF Parser.

3.2 File Descriptions

The system has some files that are persistent, including logging files and the outputs the system produces. The system will produce a TPTP or vampire output, which can be sent to COLORE, or the Common Logic Ontology Repository. This is done so the system can keep track of what outputs are consistent.

4 Requirements Matrix

In this section each functional requirement of the system is matched with the system component that will satisfy that requirement. The functional requirements are listed by name, number, and use case of each requirement. The system components are listed by either the script that is used in the system or the name of any tools that are accessed by the system.

System Component	Functional Requirement	Number	Use Case
parser.py	Identify Quantified Variables (and their scope and use)	FR-1	The system will identify variables in a given logical statement
parser.py	Identify Predicates	FR-2	The system will identify predicates and function symbols in a inputted logical statement
parser.py	Identify Statements from file	FR-3	The system can read a file and identify logical statements written inside
parser.py	Identify Syntactical Errors	FR-4	The system will identify syntax errors in the input CLIF file and open a debugger so they can be addressed.

clif_to_tptp.py	Take CLIF and output TPTP	FR-5	The system should be able to take a CLIF file as an input and give a TPTP file as an output
clif_to_ladr.py	Take CLIF file and produce LADR output	FR-6	The system should be able to take in a CLIF file and produce LADR output
clif_to_owl.py	Extract OWL approximation from CLIF ontology/module	FR-7	The system should be able to extract an OWL (Web Ontology Language) approximation from a CLIF ontology or module
check_consistency_new.py	Verify the logical consistency of a CLIF ontology or module	FR-8	The system shall be able to verify the logical consistency of a CLIF ontology/module
Theorem provers accessed by system: Prover9 Vampire	Prove theorems that encode intended consequences (e.g. properties of concepts and relations) of ontologies/modules	FR-9	The system should have theorem proving capabilities that can encode the intended consequences of ontologies/modules given to the system. Examples of intended consequences are properties of concepts and relations or competency questions.

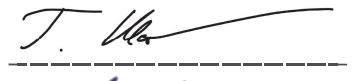
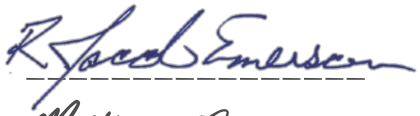
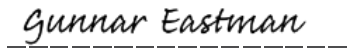
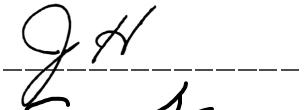
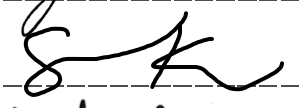
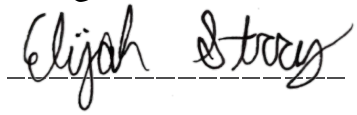
Fig. 5: Requirement Matrix of CLIF Parser: In this matrix, functional requirements are enumerated and the components responsible for completion of the requirements are described.

Appendix A

This appendix details the expectations that RCD shall uphold to the client upon completion of this document, and how future changes to this document shall be made.

RCD and the client, upon the signing of the document, are agreeing that this SDD contains a compilation of the architecture necessary for the CLIF Parser. RCD and the client agree that this architecture is to be developed over the course of the Fall 2022 and Spring 2023 University of Maine semesters. The team, RCD, agrees that this architecture is meant to be agile and flexible in nature, so if the need arises, the requirements may change in accordance with the client's wishes.

Any changes made to this document must be approved by all members of RCD and the client via signatures to an additional appendix wherein the changes are enumerated and detailed. Changes to this document include, but are not limited to, the shifting of architectural design as to be in accordance with the Capstone requirements for the University of Maine course this project is managed through. The signing of this appendix consents all members of RCD and the client that this structure of implementing changes is acceptable.

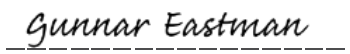
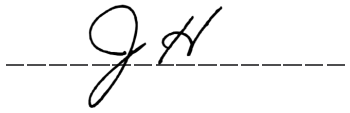
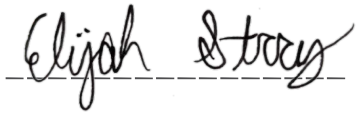
Name:	Signature:	Date:
Torsten Hahmann		11/09/22
Jake Emerson		11/09/22
Matthew Brown		11/9/22
Gunnar Eastman		11/10/22
Jesiah Harris		11/10/22
Shea Keegan		11/09/22
Eli Story		11/09/22

Customer Comments:

Appendix B

This appendix will contain the agreement that all members of RCD have read and consent to the document in its entirety.

Through signing this appendix, we, as the members of RCD, agree that we have reviewed this document fully, we agree to the formatting of this document, and agree to the content that is located within this SDD. Each member of RCD may have minor disagreements with certain parts of this document, though by signing below, we agree that there are not any major points of contention within this SRS. We have all agreed to these terms and placed our signatures below.

Name:	Signature:	Date:
Matthew Brown Comments:		11/9/22
Gunnar Eastman Comments:		11/10/22
Jesiah Harris Comments:		11/10/22
Shea Keegan Comments:		11/09/22
Eli Story Comments:		11/09/22

Appendix C

This appendix will outline the approximate contributions of each of the team members of RCD to the completion of this SDD.

Matthew Brown

- Created the Architecture Design Diagram and wrote the description
- Formatted the document
- Worked on all sections
- Contributed 35% of the document

Gunnar Eastman

- Created the document
- Created the Decomposition Diagram and wrote the description
- Worked on the entire document
- Contributed 35% of the document

Jesiah Harris

- Wrote the requirement matrix
- Worked on §4
- Contributed 20% of the document

Shea Keegan

- Wrote the Persistent Data Description
- Worked on §3
- Contributed 10% of the document

Eli Story

- Sister team review
- Did not contribute to the SDD