

pyTLEX 1.0 + jTLEX 3.0: Translation and Visualization

Students: Jared Hummer

Mentors: Mark Finlayson, Mustafa Ocal, Florida International University

Instructor/Faculty: Dr. Masoud Sadjadi, Florida International University

PROBLEM

Qualitative temporal graphs that are derived from annotations on text (TimeML annotations) explicitly reveal only the partial orderings of events and times. For many purposes, however, a total ordering (a timeline) is more useful.

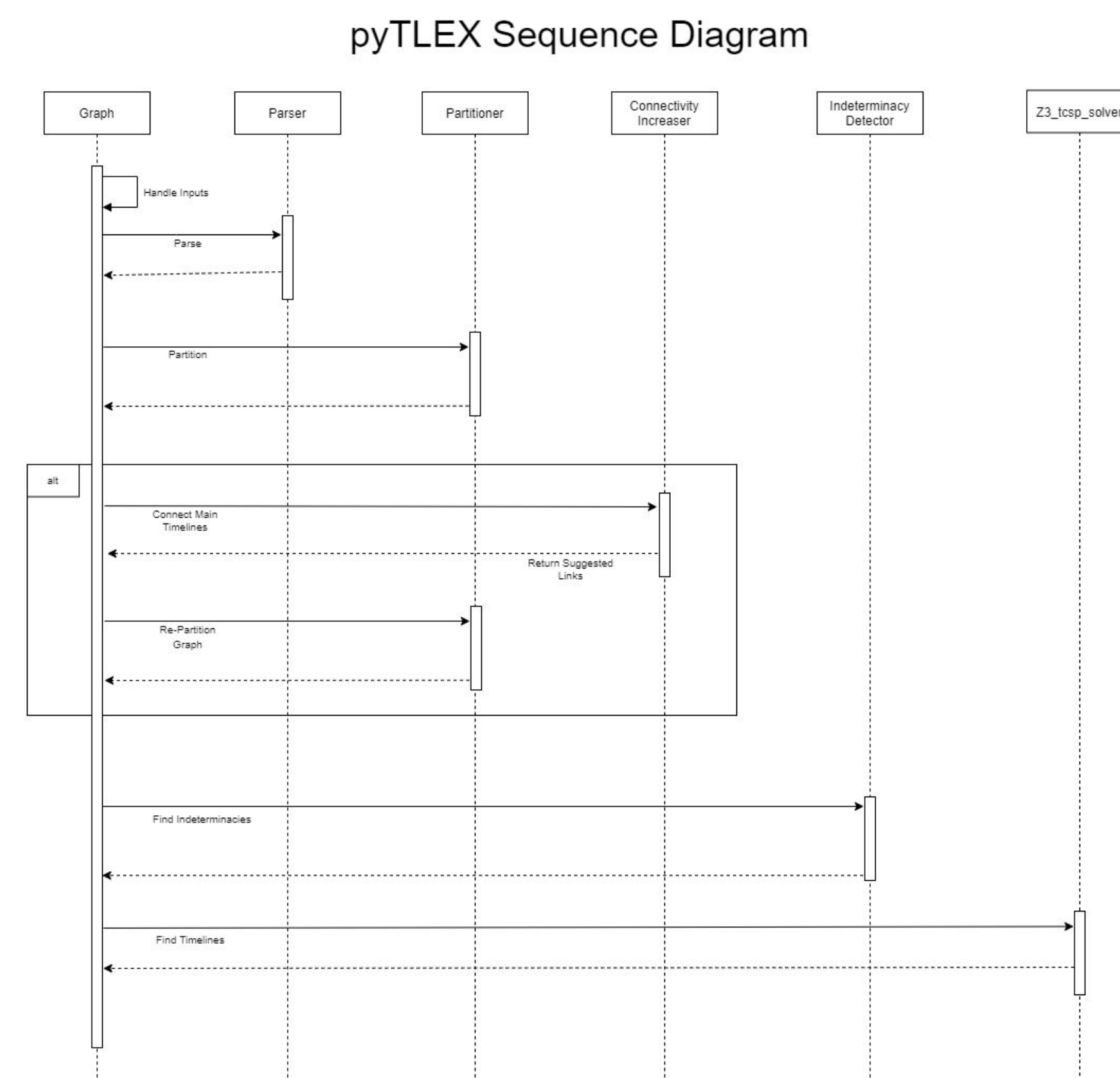
Unfortunately, timelines are rarely explicit in text, and usually cannot be extracted directly. The TLEX algorithm specifies an exact, end-to-end solution, to the task of extracting multiple timelines from TimeML annotated texts. TLEX transforms TimeML annotations into a collection of main and subordinated timelines arranged into a trunk-and-branch style structure.

jTLEX offered the solution proposed by the TLEX algorithm into an easy-to-use Java library. With it, a Web application called **TLEX: Timeline Extraction**, was created where annotated text from TimeML files or the files themselves can be uploaded and analyzed.

TLEX, with its two components; TLEX backend and TLEX frontend, aids researchers in the field by implementing a novel solution to the problem of visualizing temporal graphs.

pyTLEX also offers this solution, but through an easy-to-use Python library. This expands past the limitations of the Java language, and allows for more development using the TLEX algorithm.

SEQUENCE DIAGRAM



SUMMARY

TLEX: Timeline Extraction handles graphic displaying of all the elements obtained from the timeline extraction task using **jTLEX**. Improvements to the visualization has made the timelines easier to navigate, view, and modify.

The **pyTLEX** library is a successful translation of **jTLEX**, with improvements to the approaches of several classes. With this, the **TLEX** algorithm is now accessible to python developers.

ACKNOWLEDGEMENT

The material presented in this poster is based upon the work supported by the Cognac Laboratory and MITRE. We thank Mustafa Ocal and Dr. Mark Finlayson for their assistance and mentorship that we received throughout the senior design project.

CURRENT SYSTEM

The improved version of the **TLEX** visualization tool now supports multiple links between the same nodes. In addition, a more efficient way of displaying the TimeML graph nodes has been implemented.

Timeline visualization has been improved with a refactoring of the display of timepoints. With the current system, time pairs are connected via a line, which improves the clarity and organization.

The last major improvement in **TLEX** was the addition indeterminacy pair-related functionality. These timepoints are now colored red and a list of the pairs appears as a modal.

Finally, a whole new Python Library (**pyTLEX**) has been created and tested. It is derived from the existing jTLEX library with improved and adapted functionality.

pyTLEX supports the creation of a timeML graph from timeML annotated text, a timeML file or sets of nodes and links. This graph yields information about its partitions and their timelines. Lastly indeterminacy and inconsistency detection functionalities are supported by the library.

INDETERMINACY DETECTOR

One main step of the TLEX algorithm called Indeterminacy Detection, which in jTLEX, relies on a core functionality that could not be found in any of the Python CSP solver libraries, mainly the yielding of random solutions.

Our product owner, Mustafa Ocal, created a new algorithm for detecting indeterminacies and I realized that process with an implementation using Microsoft Research's Z3-Solver library. Upon completion of this implementation, it came to light that the current version of jTLEX is incorrect in regards to its process of indeterminacy detection. Further research is being done in the COGNAC lab to check the feasibility of the new algorithm in jTLEX.

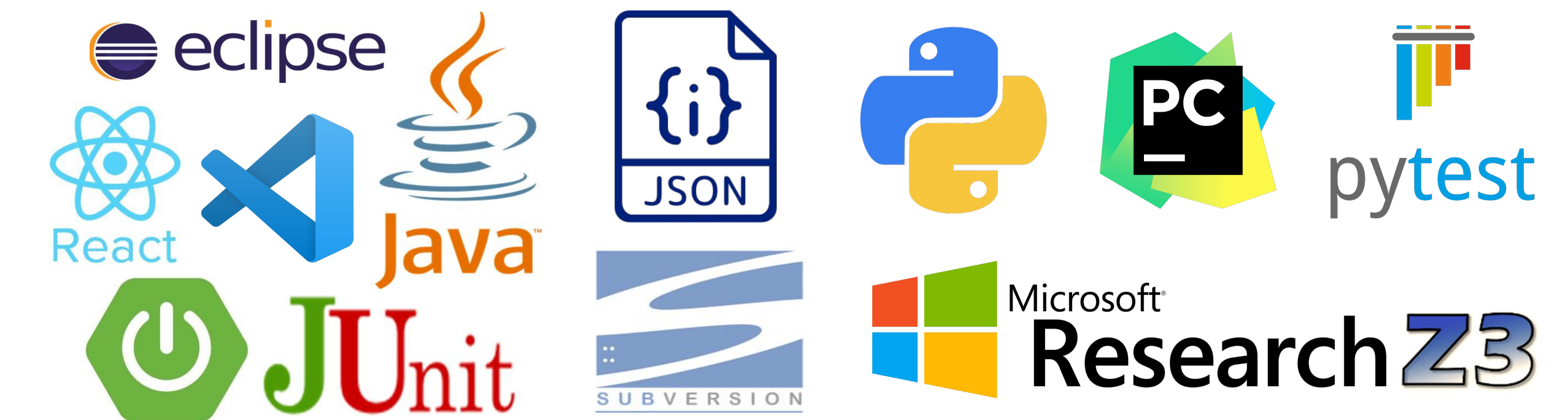
TCSP Solver

Two CSP solvers were implemented and tested, python-constraint and Z3-Solver. The latter passed all correctness and performance tests and was deemed adequate to depend on.

VERIFICATION

Purpose	Method	Preconditions	Expectation	Result
(1) Check that the pyTLEX TimeMLParser catches all annotations	Count the number of annotations and compare to the number of each object returned	Valid TimeML File	All Objects Caught	Equal Count, Successful
(2) Conduct a System Test of pyTLEX	Count the number of each individual object from jTLEX and compare that to the output from pyTLEX	Valid TimeML File	All Objects Caught	Equal Counts, Successful
(3) Show that the pyTLEX output matches jTLEX	Run both jTLEX and pyTLEX to receive json output	Valid TimeML File	Equivalent JSON output	Equivalent JSON Output Received
(4) Assess the changes to TLEX Visualization	Run both, the upload and the paste option of the tool to see if the output matches previous versions	Valid TimeML File	Graph displays the same as previous versions	Graph displays correctly like past versions

IMPLEMENTATION

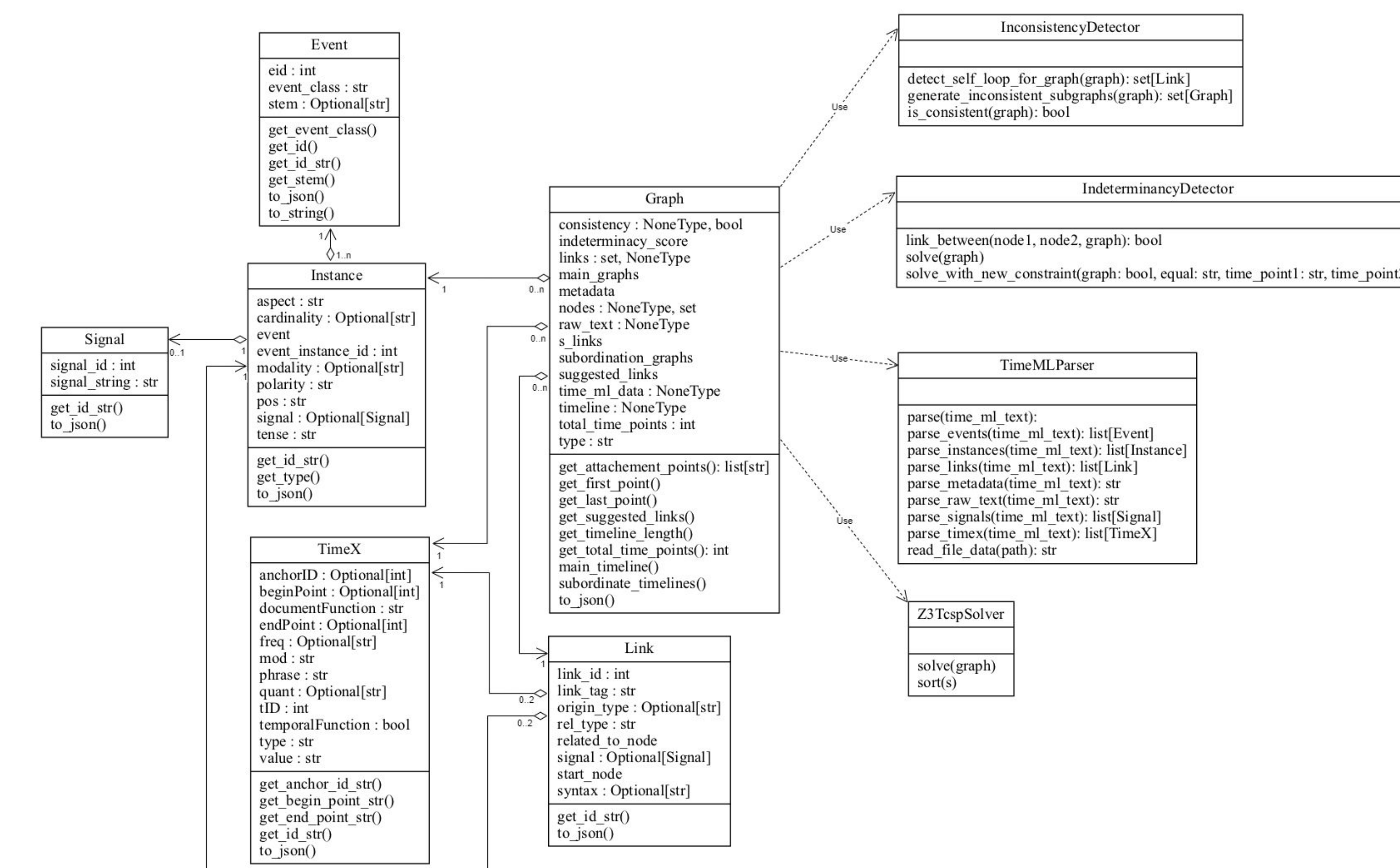


The **TLEX** backend was implemented using the Java based Spring Boot framework in the IntelliJ Integrated Development Environment. The **TLEX** frontend was developed using ReactJS in the Visual Studio Code Integrated Development Environment.

TLEX: Timeline Extraction uses the **jTLEX** library constructed in Java using the Eclipse Integrated Development Environment and tested using the JUnit testing framework. The **pyTLEX** library written in Python using Pycharm and tested using Pytest.

To effectively support team collaboration, we managed versions of the web application, **jTLEX** library, and **pyTLEX** library with subversion VCS.

OBJECT DESIGN



REQUIREMENTS

- Java
- Gradle >= 7.1
- JDK >= 11.0.1
- Node.js >= 14.17.0
- npm >= 6.14.13
- z3-solver >= 4.8.12.0
- pytest >= 3.4.2 (testing only)
- pytest_cov >= 2.5.1 (testing only)