Datastores for Argumentation Data

Simon Wells[1][0000−0003−4512−7868]

Edinburgh Napier University, Edinburgh, United Kingdom.
s.wells@napier.ac.uk
http://www.simonwells.org

Abstract. This paper introduces SADFace, a simple argument description format, and ArgDB, a datastore for managing datasets of SADface documents, in the wider context of a nascent effort to develop an Open Argumentation platform.

Keywords: Argumentation Datasets · Argumentation Tools · Argumentation Software · Open Source Software · Open Development Platforms.

1 Introduction

In recent years, there has been a gentle turn towards a more data-driven approach to argumentation research and whilst research in argumentation theory has always been rooted in real world practise, studying examples of actual argumentative behaviour amongst people, the increasing use of digital machinery to process and engage in argument is leading to the creation of, and exploitation of, ever larger argumentative datasets. A problem arises when a software developer begins to develop tools that are aimed at exploiting, facilitating, capturing, or analysing online argumentation; they often have to start from scratch, building on the published research output. This is the traditional mode of exploitation of research findings, but it represents an opportunity whereby, if a little more effort is made to enable the development of public argumentation systems, then a variety of rich sources of argumentative data might be created which can in turn be utilised in more data-intensive argumentation research such as Argument Mining[5]. This paper reports on an ongoing and open-ended effort to provide a set of free, open-source, and open-development, tools, libraries, and formats that can underpin future generations of digital argumentation software. The core of the system reported here is a format for describing argument data and a datastore for holding that data. These are introduced in the context of argument analysis, using the MonkeyPuzzle tool, but actually form core technologies for the Open Argumentation PLatform (OAPL)[1] [14].

Whilst much argumentation software has been developed over the years, supporting a wide range of argumentation-related tasks (including Araucaria [9], Arvina [4], Carneades [3], Rationale [1] to name but a few), considerably less

[1] http://www.openargumentation.org

CMNA'20 - F. Grasso, N. Green, J. Schneider, S. Wells (Eds) - 8th September 2020
Copyright ©2020 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR-WS.org/Vol-2669/paper4.pdf

effort has gone into producing a sustainable eco-system of tools that can work together, can support a distinct and identifiable set of argumentation-oriented workflows, and which can bring applied argumentation theory out of the lab and into the wider software and computational communities. Some efforts have occurred to promote some cohesion between software, most notably the Argument Interchange Format (AIF)[2], however this hasn't seen the kind of community uptake and tooling provision that is necessary for it to fulfill a role as the interface to argument data for the wider software community. Rather, the AIF has arguably developed into a useful and powerful theoretical and ontological model but lacks the ease of use and immediacy that characterises many tools, especially in the Web arena, that see significant and wide-spread uptake. An example of this occurring in data representation would be comparing the power and extensibility, and consequent challenge to ease of use, of RDF and XML languages and technologies, versus JSON which has arguably become the standard for data interchange on the Web.

It is important to note that whilst proposals are made herein for particular tools to achieve specific aims, e.g. SADFace to describe structured argument and ArgDB to store those structures, this is not meant to occur in an exclusive fashion. An ideal outcome is that there is a healthy ecosystem of inter-operable argumentation software, featuring a multitude of ways to achieve the various goals that a developer or researcher might set and that it is easy for any one, or even for all, of the tools in a given toolchain to be exchanged for alternatives. There is some healthy development in this area, particularly in the tooling around ArgDown[2] [11] and the continued development of the Carneades argumentation system. This work differs in somewhat to each of these, aiming to provide a more comprehensive set of tools than ArgDown, and providing a less integrated and more flexible toolset than Carneades, but coexisting with each. The remainder of this paper is structured as follows: In sections 2 and 3 the underpinning technologies, SADFace and ArgDB, respectively for this architecture, are introduced. In section 4 a brief overview of an argument analysis pipeline is explored. Finally, in section 5, some directions for future work, and the wider context of an open argumentation platform, are discussed.

2 SADFace

The Simple Argument Description Format (SADFace) is a JSON[3] language for describing and sharing structured argument data. JSON is a lightweight data interchange format that is generally considered to be more human-readable than competitor formats like XML or RDF. This means that SADFace can easily be manipulated by a person using a plain-text editor. However, to do so misses out on the additional tooling that supports the SADFace language. In the remainder of this section, we will give an overview of the SADFace language in section 2.1 before describing the supporting tools and library implementations in section 2.2,

[2] https://argdown.org/
[3] https://www.json.org/json-en.html

before briefly discussing the role of SADFace in relation to the AIF in section 2.3. Detailed information regarding SADFace is available from the Git repository[4].

2.1 SADFace Language

SADFace is a simple JSON based Argument description format, software library, and supporting tools to enable developers and researchers to describe arguments, capturing content and associated metadata, and to then easily reuse their data. The goal is to make it as easy as possible to incorporate argument data into modern software. There are a number of argument description and markup languages, for example, the Argument Markup Language (AML) and the Argument Interchange Format (AIF). Both have their advantages and disadvantages, AML is a simple tree-based format, but is challenging to parse, and requires an XML toolchain, something that is falling out of favour amongst main-stream web toolchains. AIF on the other hand provides an excellent, extensible high-level ontology for representing arguments, dialogues, schemes, and many peripherally related concepts. However with AIF that flexibility has the price of opaque terminology and complexity, both in terms of underlying model and in terms of the required RDF and XML toolchains. Where both of these formats fail is in the documentation, support, and maturity of libraries for users who want to work with them. It is a big, perhaps prohibitive, requirement for a developer to start their adoption of a format by first having to write a parser or library for it. A more preferable set of circumstances is for the developer to be able to merely import a pre-made and tested library for their chosen language. SADFace seeks to address some perceivable drawbacks of other formats. It has a simple underlying but extensible model that is compatible with core AIF concepts, clear extension points for domain specific analysis and representation tasks, tooling to support import from other formats, e.g. AML and AIF, an open source canonical implementation maintained in a public Git repository, documentation, supporting tools for creation, editing, and manipulation, liberal (GPL3) licensing, and an open development model in which anyone is free to contribute enhancements or to fork their own development branch. A developer should be able to adopt SADFace and start describing arguments or using those descriptions, really easily. Parsing a SADFace document into a Javascript application should not require any special tools, it is just JSON. The structure of the format has been designed to align with a straightforward model of argument structure, defined in such a way as to align with most of the intuitions that an everyday understanding of argument will include, whilst still supporting more advanced features.

In SADFace, arguments are constructed from statements (or strings) that capture a "claim". These statements are called "Argument Atoms", or just "atoms". A simple argument is a collection of such atoms that are linked together. For a simple argument one of the atoms is a conclusion, and the other atoms in that argument are premises. The exact way that premises support a

[4] https://github.com/Open-Argumentation/SADFace

conclusion is captured by the idea of argumentation schemes[12]. An argumentation scheme describes a stereotypical pattern of argumentative reasoning, so different arguments can be categorised into different types. In SADFace atom nodes are not directly linked to each other but are linked via an "Argumentation Scheme Node", or "scheme" node. Whilst we often characterise arguments in terms of a directed graph, with the premises leading to a conclusion, which in turn might act as a premise in a further argument, with some other conclusion, SADFace edges are bi-directional, enabling a programmer to move through the argument graph easily in either direction. A SADFace document contains a set of nodes, a set of edges, and metadata. Edges are simple, they have an ID so that the instance of the edge can be uniquely identified from all other edges, and they also have a source and a target. The source and target in an edge are both IDs of nodes, i.e. the ID of the node that this edge goes from, the source ID, and the ID of the node that this edge goes to, the target ID. Nodes are slightly more complicated, currently there are two types of node, atom nodes and scheme nodes. Atoms and schemes are connected using the aforementioned edges. Metadata captures data associated with the document, who analysed, when this happened, a title, additional notes, etc. An optional element of SADFace is the resources section which records all of the sources that the argument has been analysed from. This enables a single SADFace document to reference multiple underlying originating sources of content and opens the way for an analysis to capture an entire target domain, rather than a single source text. An example SADFace document is shown in Fig. 1. An important aspect of the SADFace design is the defined extension points so that end-users add their own metadata, to suit their own problem domain, in namespaced sections (where the "core" namespace is reserved for SADFace usage) to which they can add their own metadata. Such sections are referred to as 'regulated' sections. Additionally, the validation methods provided in the SADFace libraries work on the following two principles: firstly, the required sections must be present but additional section will be ignored, and secondly, all required sections and all regulated sections must be correctly formed.

2.2 SADFace Implementation & Tooling

There are currently two library implementations supporting SADFace, a Python 3 and a JavaScript implementation which are developed primarily to provide native library APIs within their respective languages. The JavaScript implementation is intended to enable web apps, running in the browser, or hosted in Node.JS, to read, write, and verify SADFace documents. The Python implementation is more mature than the JavaScript implementation and provides supporting tools for interacting with SADFace documents beyond merely creating and updating them. For example, the Python implementation additionally support upgrading and downgrading versions of SADFace as the format develops, enhanced validation, pretty-printing, as well as import and export to other formats like AML, AIF, and Dot/GraphViz. In addition to use as a library, the Python implementation has other modes of usage including a scriptable command line interface

```
{"edges": [
        {
            "id": "3df54ae1-fa41-4ac7-85d5-4badee39215b",
            "source_id": "70447169-9264-41dc-b8e9-50523f8368c1",
            "target_id": "ae3f0c7f-9f69-4cab-9db3-3b9c46f56e09"
        },
        ...
    ],
    "metadata": {
        "core": {
            "analyst_email": "siwells@gmail.com",
            "analyst_name": "Simon Wells",
            "created": "2019-04-22T23:52:30",
            "description": "An example SADFace document showing an argument
                analysis of the Hangback cycle safety campaign from the
                STCD corpora.",
            "edited": "2019-04-22T23:52:30",
            "id": "42e56df7-4074-40d8-8ea1-4fca5321dd31",
            "notes": "This is incomplete because the analysis in Pangbourne
            \& Wells (2018) has much more argumenative content.",
            "title": "Hangback Example",
            "version": "0.2"
        }
    },
    "nodes": [
        {
            "id": "ae3f0c7f-9f69-4cab-9db3-3b9c46f56e09",
            "metadata": {},
            "sources": [],
            "text": "The 'Hang Back' campaign video should not have been
                published, and should be withdrawn.",
            "type": "atom"
        },
        ...
    ],
    "resources": []}
```


Fig. 1. A simple example of a SADFace document that illustrates the four main blocks that comprise a SADFace document; the edge set, the nodes set, the resources set, and the metadata. To conserve space, multiple edge and nodes instances have been elided and replaced with ....

that exposes most core API functions from the SADFace library. This is useful for incorporating SADFace within a pipeline or shell script, for example, to take an AML document, convert it to SADFace, then to export the resulting file to a DOT representation and pipe it to GraphViz, ultimately yielding an image file for visualisation. Furthermore, there is a Read-Evaluate-Print-Loop (REPL) that enables users to interactively construct and manipulate SADFace documents from within a text oriented shell interface.

2.3 SADFace & The Argument Interchange Format

The question, 'why not just use the AIF?' is appropriate and will now be addressed. AIF is an excellent, general-purpose, upper ontology for representing and describing argumentation concepts in an extensible fashion. The AIF has enabled argumentation researchers to distinguish a variety of argument-related concepts and to describe those concepts in technical detail sufficient for computational exploitation. For example, describing concepts related to Argumentation Schemes [7], Dialogue [10], and social interaction [6]. However, whilst theoretical extensions to the AIF have continued to develop, and the AIF provides a useful *lingua franca* for effective communication and understanding between researchers, this has not been matched by uptake amongst software developers in subsequent years, despite some effort to present a cohesive online presence[5]. For example, whilst the web has developed into a *de facto* environment for discussion, the infrastructure for exploiting and reusing the content of those discussions, has not kept pace. The AIF does not have a canonical implementation, for example, a software library, that enables developers to exploit the rich theoretical knowledge captured therein. SADFace has been developed to be consistent with AIF design principles, such as the graph based core structure and restriction on edges carrying data, but a key design decision was to be consistent, understandable, and build around a small, solid extensible core that is reified in a canonical software implementation, and which is in turn underpinned by testing and documentation.

3 ArgDB

The Argument Database (ArgDB) is a datastore and supporting software for collecting, manipulating, interacting with, and reusing, analysed and structured argumentation data described using the SADFace format. More specifically, ArgDB is a set of Python tools for creating, managing, and searching multiple datasets of SADFace documents. Python was selected because it is a mature language that is widely used and is particularly prevalent in server-side Web development, in data science, and in modern, data-driven AI research. As a result the use of Python should reduce the barrier to entry for adoption of ArgDB in those areas. The idea is that those using argumentative data may need to keep some datasets

[5] http://www.argumentinterchange.org/

separate from others, for example, if working on different funded projects, or because of the nature of the data requires heightened security or privacy steps to be taken such as when working with private, personal, medical, or legal argument data. A core design decision is support for multiple, separate databases that can all be managed from a single ArgDB instance. This enables a user to store cohesive datasets separate from each other, if collating everything together is undesirable, or else to mix argument documents together as required by their own goals. Technically, ArgDB provides a set of argumentation task oriented interfaces to a pre-existing, underlying database technology. Presently this technology is the Apache CouchDB[6] a JSON document store that provides map-reduce based indexing and search capabilities together with an HTTP/JSON API and replication and synchronisation capabilities. Whilst there are other datastores that provide an overlapping set of features which might have sufficed for present purposes, out-of-the-box CouchDB provides all of the functionality necessary for the current ArgDB, as well as a pathway towards future features. For example, one very desirable feature would be to enable researchers and groups to easily share and replicate argumentative datasets, a feature that can be built into ArgDB by exploiting CouchDB's existing replication and synchronisation features. Detailed information regarding SADFace installation and usage is available from the project's Git repository[7].

3.1 ArgDB Interfaces

The ArgDB tools are implemented in Python 3, support a variety of ways to interact with the ArgDB, and are designed to support the construction of a variety of workflows. Current interfaces include an application programming interface, a command line interface (CLI), a read-evaluate-print loop (REPL), and a graphical user interface (GUI). A single Python implementation hosts the core of the ArgDB functionality, which provides an API for interacting with the underlying CouchDB software. This API is intended for integration of ArgDB functionality within end-users own software applications. This API is also used by the CLI, REPL, and GUI interfaces to access ArgDB functionality. The CLI provides a scribtable interface, enabling single command line invocations, or more extensive scripts to create and delete databases, to verify, add, update, and delete, SADFace documents within those databases, and to search the contents of the stored documents, either withing a single database or across multiple databases. The GUI utilises a web view facilitated by the Python PyWebView[8] project. This enables an interface based upon the core web technologies, HTML, CSS, and JavaScript to be provided, in a cross-platform fashion, to multiple operating systems. The web view provides an interface to the native installed web browser and enables the ArgDB GUI to appear as a local graphical application. The current GUI interface is shown in Fig. 2 and is currently, primarily, focussed upon exposing the search facility to users.

[6] https://couchdb.apache.org/
[7] https://github.com/Open-Argumentation/ArgDB
[8] https://pywebview.flowrl.com/

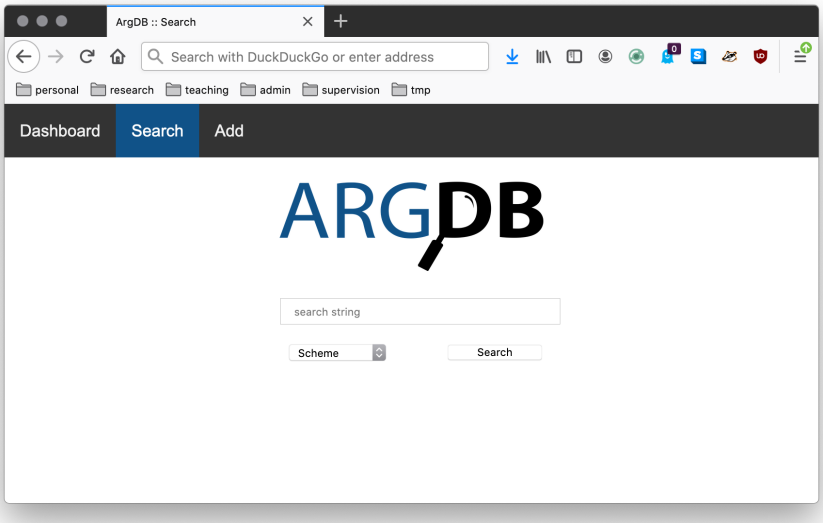

Fig. 2. The GUI for ArgDB displaying the search box in the center above a drop-down widget for restricting search to a specific SADFace document key, for example, analyst ID, scheme name, or date range.

4 Case Study: Using MonkeyPuzzle to construct a persistent collection of analysed argumentative data

In order to demonstrate ArgDB in earnest requires a source of argumentation data. The easiest method is to generate newly analysed arguments, represent these analyses using SADFace, and to store the result in ArgDB. This is merely one of the core workflows that is currently supported but is not the end-goal[9]. MonkeyPuzzle [15] is a browser-based, open-source, argument analysis web-interface. It currently runs either standalone or hosted on a web server. Basic interaction is in the style familiar from Araucaria, text is selected from a resource pane and used to create a new node in the diagram. Nodes are then linked together via scheme nodes to construct arguments in which the arrow directed out from a scheme indicates a conclusion, and arrows directed towards a scheme indicate premises. Users can load an argumentative text into MonkeyPuzzle and analyse it in the fashion of Rationale or Araucaria. The analysed argument can be exported to a SADFace document using the JavaScript SADFace library and imported into an ArgDB database. This structured argument document can then be retrieved, updated, or deleted from the resulting dataset but is persisted ready for retrieval in other contexts.

[9] See the future work described in section 5 for an indication of the kinds of workflows that are in active development

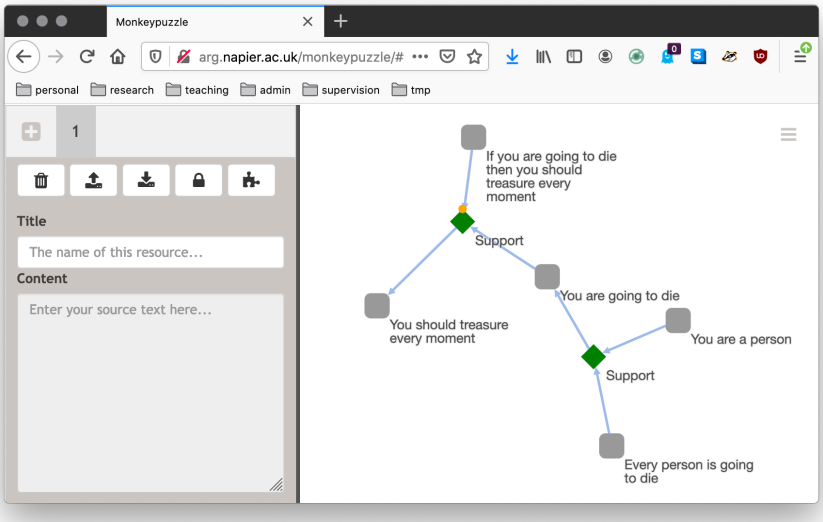

Fig. 3. The user interface of MonkeyPuzzle displaying a demo argument analysis.

5 Conclusions & Future Work

To conclude, we have presented two technologies, SADFace and ArgDB, that work together to provide a data storage platform for structured argumentation data. One strand of future work will involve continuing to enhance the core SADFace technology whilst exposing any new functionality via ArgDB. A second strand of work places these two technologies at the core of a wider 'Open Argumentation PLatform' (OAPL). This is a nascent effort to stimulate the development of a set of open source argumentation technologies that (1) work together to provide a coherent set of argument oriented toolchains and workflows (to support (a) automated reasoning using the ALIAS tool and library [16], (b) argument mining [13], and (c) dialogical interaction based upon the Dialogue Game Description Language (DGDL) [17]), (2) can form the basis of new research directions, (3) have a long-term, open development plan, and (4) which can be integrated more easily into the wider world of computing, perhaps to help fulfill some of the promise of an Argument Web[8].

References

1. ter Berg, T., van Gelder, T., Patterson, F., Teppema, S.: Critical Thinking: Reasoning and Communicating with Rationale. CreateSpace Independent Publishing Platform (2013)

2. Chesnevar, C., McGinnis, J., Modgil, S., Rahwan, I., Reed, C., Simari, G., South, M., Vreeswijk, G., Willmott, S.: Towards an Argument Interchange Format. Knowledge Engineering Review **21**(4), 293–316 (2006)
3. Gordon, T.F.: Visualizing Carneades Argument Graphs. Law, Probability and Risk **6**(1–4), 109–117 (2007)
4. Lawrence, J., Bex, F., Reed, C.: Dialogues on the Argument Web: Mixed Initiative Argumentation with Arvina. In: Proceedings of the 4th International Conference on Computational Models of Argument (COMMA 2012). pp. 513–514. IOS Press, Vienna (2012)
5. Lawrence, J., Reed, C.: Argument Mining: A Survey. Computational Linguistics **45**(4), 765–818 (2019)
6. Lawrence, J., Snaith, M., Konat, B., Budzynska, K., Reed, C.: Debating Technology for Dialogical Argument: Sensemaking, Engagement, and Analytics. ACM Transactions on Internet Technologies **17**(3), 1–23 (2017)
7. Rahwan, I., Banihashemi, B., Reed, C., Walton, D., Abdallah, S.: Representing and Classifying Arguments on the Semantic Web. The Knowledge Engineering Review. **26**(4), 487–511 (2010)
8. Rahwan, I., Zablith, F., Reed, C.: Laying the Foundations for a World Wide Argument Web. Artificial Intelligence **171**, 897–921 (2007)
9. Reed, C., Rowe, G.: Araucaria: Software for Puzzles in Argument Diagramming and XML. Tech. rep., University Of Dundee (2001)
10. Reed, C., Wells, S., Rowe, G.W.A., Devereux, J.: Aif+: Dialogue in the Argument Interchange Format. In: Proceedings of the 2nd International Conference on Computational Models of Argument (COMMA 2008) (2008)
11. Voigt, C.: Argdown and the Stacked Masonry Layout: Two User Interfaces for Non-Expert Users. In: Proceedings of the 2014 conference on Computational Models of Argument: Proceedings of COMMA 2014. pp. 483–484. IOS Press, Amsterdam (2014)
12. Walton, D.N.: Argumentation Schemes for Presumptive Reasoning. Lawrence Erlbaum Associates (1996)
13. Wells, S.: Argument Mining: Was Ist Das? In: Proceedings of the 14th International Workshop on Computational Models of Natural Argument (CMNA14) (2014)
14. Wells, S.: The Open Argumentation PLatform (oapl). In: Proceedings of the 8th International Conference on Computational Models of Argument (COMMA 2020) (2017)
15. Wells, S., Douglas, J.: Monkeypuzzle: Towards Next Generation, Free & Open-Source, Argument Analysis Tools. In: Proceedings of the 17th International Workshop on Computational Models of Natural Argument (CMNA17) (2017)
16. Wells, S., La Greca, R.: Introducing ALIAS. In: Proceedings of the 15th International Workshop on Computational Models of Natural Argument (CMNA15) (2015)
17. Wells, S., Reed, C.: A Domain Specific Language for Describing Diverse Systems of Dialogue. Journal of Applied Logic **10**(4), 309–329 (2012)