Pipelining Argumentation Technologies

Mark SNAITH, Joseph DEVEREUX, John LAWRENCE and Chris REED
School of Computing, University of Dundee, Dundee, DD1 4HN, UK

Abstract. Software tools for working with argument generally exist as large systems that wrap their entire feature set in the application as a whole. This approach, while perfectly valid, can result in users having to use small parts of multiple systems to carry out a specific task. In this paper, we present a series of web services, that each encapsulate small pieces of functionality for working with argument. We then demonstrate two example systems built by connecting these services in the form of a UNIX-style pipeline.

Keywords. Argumentation Engine, Dung, Dungine, Dung-O-Matic, Web Services

1. Introduction

Argumentation tools such as Araucaria [9], Rationale [12], AVERS [1] and Cohere [10] all provide distinct features for working with argument. However, each system wraps its entire feature set in the application as a whole, meaning individual components cannot be used independently. As a result, users may find themselves having to use multiple systems to carry out a specific task.

We present here a solution to this problem, that involves breaking down argumentation tools into small components, deploying these components as web services, then constructing UNIX-style pipelines to link them together as one large system, with specific functionality.

The advantages of this approach are that components can be used in multiple systems that require overlapping functionality, but only one instance of each component is required (on a remote web server). Additionally, adopting a web services approach allows existing, legacy systems to interact with new technologies, and vice versa (see section 4.2).

2. Web services

In creating our web services, we decided to use the REST (REpresentational State Transfer) architecture [7], because this is simpler to interact with than protocol-based approaches such as SOAP (Simple Object Access Protocol).

In addition, REST allows for simpler construction of services, which can be written in any language that allows parameters to be posted to it. We illustrate this point in the deployment of our services; most were written in Java using the RESTlet framework[1], however one was written in PHP.

[1] `http://www.restlet.org/`

2.1. Dungine

Dungine is a Dung reasoner, for the computation of arguments in the grounded extension and every preferred extension of a given framework [11]. It exists as a Java library, which we have wrapped in a RESTful interface.

The interface both accepts and returns an adapted RDF reification of the Argument Interchange Format (AIF) [4] (see section 2.5). Additionally, the type of semantics required (i.e. grounded or preferred) is indicated by a boolean parameter. The returned AIF contains labellings to indicate argument acceptability under the chosen semantics.

2.2. Dung-O-Matic

Dung-O-Matic is a Dung reasoner that identifies several extensions of an argumentation framework:

- the admissible sets [13]
- the preferred extensions
- the grounded extension [5]
- the stable extensions
- the semi-stable extensions [2,3]
- the ideal extension [6]
- the eager extension [3]

It can also answer any corresponding question about any argument in the framework, such as whether or not it is in the grounded extension.

This service accepts and returns the same adapted RDF reification of the AIF as the Dungine service, with the returned AIF containing labellings to indicate argument acceptability for all semantics.

2.3. ArgDB

ArgDB is an AIF reification which allows for the storage and retrieval of argument data in multiple forms. It consists of two main components:

- A web service interface which allows for the addition and retrieval of elements corresponding to the top level concepts of the AIF ontology. For example nodes, edges and schemes can all be posted individually to ArgDB. Received elements are checked for validity and that they do not currently exist in the database before adding them. The web service also offers the ability to query an ArgDB installation for the arguments it contains.
- An underlying database for the storage of each of these elements.

ArgDB currently uses a MySQL database, though it is designed to support a wide range of databases with the web service abstracting the database interaction

Additional components allow for addition and retrieval of AIF-DF directly from ArgDB through a RESTful interface, though these are not a core part of the database solution and could easily be replaced by components to handle input and output of other AIF reifications.

2.4. AML to AIF

Our AML to AIF translation module accepts Argument Markup Language (AML), as used by Araucaria [9], and translates it into AIF-DF, for insertion into ArgDB. AML "propositions" are translated into AIF I-Nodes, while AIF S-Nodes are created based on the edges between propositions (expressed in AML as "Convergent Argument" (CA) and "Linked Argument" (LA)).

This was developed to allow users to save arguments analysed in Araucaria (which does not currently support AIF) into ArgDB.

2.5. Dot to AIF/AIF to Dot

A further two translation services allow for translation between Dot language ("Dot") and AIF-DF, and vice versa.

Dot is a simple language that allows graphs to be expressed in plain text[2]. We see this as an ideal way of constructing Dung-style argument frameworks [5], which can be represented as directed graphs.

While there is presently no support for argument frameworks in the AIF, we have adapted it slightly to provide this. This was done to illustrate a distinct advantage of our pipelining approach: if and when official support for argument frameworks is added to the AIF, modifying this small component will be considerably easier than modifying a full system.

2.6. AVERS to AIF

Our final web service is still under development, but we include it here to illustrate a further advantage to our approach.

This service will translate the data used by AVERS [1] into AIF-DF. AVERS uses a version of Dot language for representation of argument, which is slightly different to the version accepted by the Dot to AIF translation service. Specifically, AVERS encodes various visual conventions in the Dot representation (ellipses correspond to inferences, etc.). Those conventions all correspond to ontological categories in the AIF specification, so constructing the mapping from AVERS Dot output to AIF-DF is straightforward because there is a common ontological backdrop provided by the AIF. Implementation is similarly straightforward, because the Dot-AIF parser is trivially extended to be sensitive to the AIF conventions implicit in AVERS Dot.

This shows that modularity enhances incremental development – by having small components with limited functionality, it is much easier to adapt those components to fulfil a slightly different goal.

3. Pipelining

It is our goal to create pipelines out of our web services, where the output from one is sent as the input to another, and so on. The concept of pipelining is not new; UNIX

[2] http://www.graphviz.org/doc/info/lang.html

systems have offered support for this approach for many years. However, applying this to argument technology is new.

An example of pipelines emerging on the web is through *Yahoo! Pipes*[3]. A drag-and-drop interface is provided, allowing various components for manipulating data to be linked together [Figure 1].

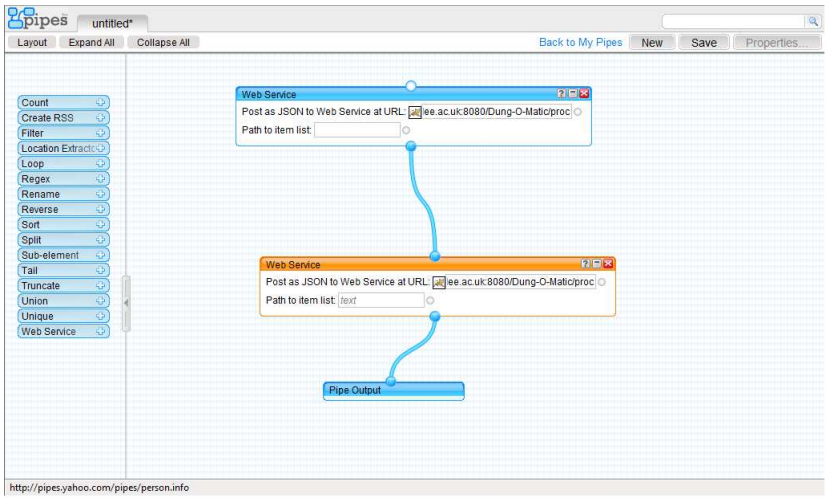

Figure 1. Yahoo! Pipes screenshot

4. Systems

We now present two systems built using different pipelines through our web services.

4.1. OVA-gen

OVA-gen is a tool for the construction and analysis of Dung-style argument frameworks [5].

It consists of a graphical ”point-and-click” interface, which sits in front of two pipelines, consisting of Dung-O-Matic and Dungine for acceptability computation, and the Dot to AIF/AIF to Dot translators. The pipeline construction can bee seen in figure 2.

The user interface was developed as part of the Online Visualisation of Argument (OVA) project, which aims to deploy easy-to-use tools for online argument visualisation[4].

A menu bar is provided at the top of the interface, allowing the user to select which semantics they wish to compute, along with which computation engine to use (for example, “Grounded (Dungine)” or “Preferred (Dung-O-Matic)”).

[3] `http://pipes.yahoo.com`
[4] `http://www.arg.dundee.ac.uk/?page_id=143`

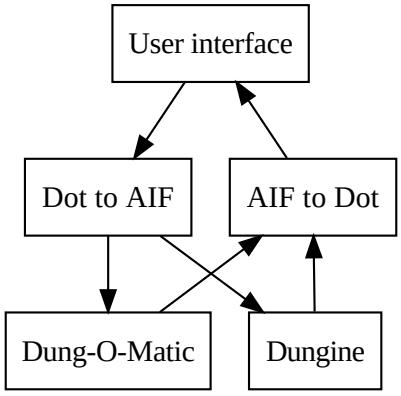

Figure 2. OVA-gen pipeline

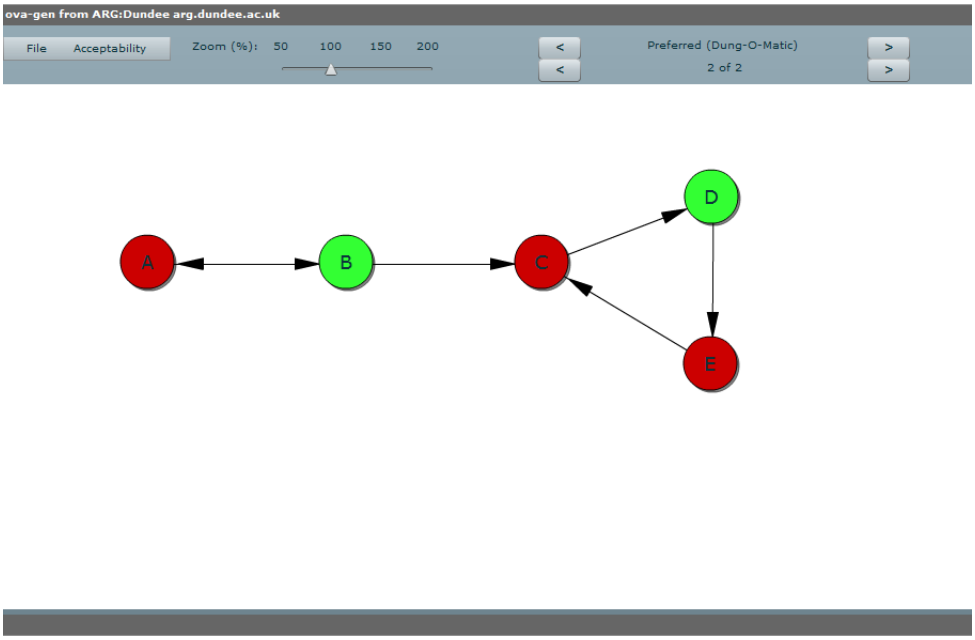

Figure 3. OVA-gen user interface

When the semantics are returned, the arguments in the framework are rendered green if they are acceptable under those semantics, or red if they are not. Under semantics that can yield more than one extension, buttons are provided for scrolling through the different extensions.

Finally, when semantics have been computed, any subsequent arguments added to the framework will immediately be rendered as acceptable. When new attacks are added, all renderings are updated to reflect the presence of that attack.

The user interface can be seen in figure 3.

4.2. AurauAIF

ArauAIF is a pipeline that takes arguments saved in the AraucariaDB in AML, translates them into AIF and inserts them into ArgDB. Its structure can be seen in figure 4.

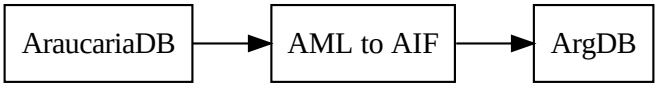

Figure 4. ArauAIF pipeline

The purpose of this pipeline is to allow users to continue using Araucaria and its connection to AraucariaDB, while also having the ability to save their analyses in AIF, into ArgDB. This benefits ArgDB and the World Wide Argument Web (WWAW)[8], because it allows Araucaria's existing userbase to seamlessly contribute to this emerging platform.

Addtionally, we have another pipeline that carries out the reverse process, translating AIF to AML, allowing AIF analyses to be loaded into Araucaria for examination and manipulation. However, this pipeline has its limitations, which are detailed in section 5.

5. Limitations & Problems

It is not our intention here to present an idealistic view of using web service pipelines for argumentation technologies. During the course of this work, we encountered several challenges, some of which remain unresolved.

Connecting our components to existing systems (such as in section 4.2) presented a problem as to how we obtain the data from the application. Our solution was to use data that was saved in the original format and convert that — so in the case of Araucaria, analyses are first saved as AML in the AraucariaDB, then translated to AIF for insertion into ArgDB.

Another issue related to the ArauAIF pipeline is that while AML is tree-based, AIF is graph-based. This is fine for translation from the former to the latter, but when we reverse the process, we find that certain analyses in AIF cannot be converted, limiting the level of interaction between Araucaria and the WWAW.

There is also the more general issue of web services only being available so long as the servers hosting them remain "live". If a server hosting a particular component (or pipeline) goes down, the functionality that it offers would no longer be available to the application employing it. The emergence of new technolgies for redundancy, such as cloud computing, could address this, but it remains an issue nonetheless[5].

6. Conclusions and future work

In this paper we have presented several web services that each encapsulate small pieces of functionality for working with argumentation. We have then demonstrated how these web services can be connected in the form of a pipeline to create larger systems.

[5] And of course, redundancy does not address the even wider issue of the client requiring an Internet connection to make use of web services.

The work presented here shows only the first steps towards a pipelining approach. Further work will be required to simplify pipeline construction; it is our intention to create a "construction toolkit" for building argumentation systems, similar in principle to that of Yahoo! pipes (section 3), but with specific components for argumentation, instead of more general tools for working with data. It would then potentially be possible to provide an open platform for developers to submit their web services for inclusion in the toolkit.

At the moment, though, we have shown how existing, reusable argumentation services can be composed into nontrivial applications in several different domains.

References

[1] S. W. Van Den Braak. Avers: An argument visualization tool for representing stories about evidence. In *In Proceedings of the 11th International Conference on Artificial Intelligence and Law*, pages 11–15. ACM Press, 2007.

[2] M. Caminada. Semi-stable semantics. In *Proceedings of the 1st International Conference on Computational Models of Argument (COMMA 2006)*, 2006.

[3] M. Caminada. Comparing two unique extension semantics for formal argumentation: Ideal and eager. In *Proceedings of BNAIC 2007*, 2007.

[4] C. Chesnevar, J. McGinnis, S. Modgil, I. Rahwan, C. Reed, G. Simari, M. South, G. Vreeswijk, and S. Willmott. Towards an argument interchange format. *The Knowledge Engineering Review*, 21(4):293–316, 2006.

[5] P. M. Dung. On the acceptability of arguments and its fundemental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77:321–357, 1995.

[6] P.M. Dung, P. Mancarella, and F. Toni. Computing ideal sceptical argumentation. *Artificial Intelligence*, 171:642–674, 2007.

[7] R.T. Fielding and R.N. Taylor. Principled design of the modern web architecture. *ACM Trans. Internet Technol.*, 2(2):115–150, 2002.

[8] I. Rahwan, F. Zablith, and C. Reed. Laying the foundations for a world wide argument web. *Artificial Intelligence*, 171:897–921, 2007.

[9] C. Reed and G. Rowe. Araucaria: software for argument analysis, diagramming and representation. *International Journal on Artificial Intelligence Tools*, 13:961–980, 2004.

[10] S. Buckingham Shum. Cohere: Towards web 2.0 argumentation. In *Proceedings of the 2nd International Conference on Computational Models of Argument (COMMA 2008)*, 2008.

[11] M. South, G. Vreeswijk, and J. Fox. Dungine: a java dung reasoner. In *Proceedings of the 2nd International Conference on Computational Models of Argument (COMMA 2008)*, 2008.

[12] T. van Gelder. The rationale for rationale. *Law, Probability and Risk*, 6(1–4):23–42, 2007.

[13] G. A. W. Vreeswijk. An algorithm to compute minimally grounded and admissible defence sets in argument systems. In *Proceedings of the 1st International Conference on Computational Models of Argument (COMMA 2006)*, 2006.