Implementing ANGELIC Designs Using Logiak

Katie Atkinson, Trevor Bench-Capon
Department of Computer Science, The University of Liverpool, UK

Tom Routen, Alejandro Sánchez
Things Prime GmbH, Basel, Switzerland. routen@mangologic.com

Stuart Whittle, Rob Williams and Catriona Wolfenden
Weightmans LLP, Liverpool, UK

Abstract. ANGELIC is a methodology for encapsulating knowledge of a body of case law. As part of system development, the ability to rapidly produce a working implementation from the design, is important for verification and refinement. Logiak is a system intended to support the development of logic programs by domain experts, and so provides an excellent environment for the rapid realisation of ANGELIC designs. We have used Logiak to realise ANGELIC designs, using both Boolean factors and factors with magnitude. We illustrate the process with both a Boolean example (Noise Induced Hearing Loss) and an example using factors with magnitude (US Trade Secrets based on CATO).

1. Introduction

The ANGELIC methodology for designing systems intended to encapsulate case law was described in [2]. In partnership with the UK law firm, Weightmans, this methodology was used to build a substantial application designed to support decisions as to whether or not claims for compensation for Noise Induced Hearing Loss (NIHL) should be contested [3]. The process of implementing the design in these projects was, however, not entirely satisfactory: the translation to Prolog, although fairly mechanical, required considerable effort and resulted in a stylistically poor program. Moreover, a custom built interface had to be produced from scratch. To address these problems we explored the use of a target implementation platform, Logiak, which would enable rapid and convenient realisation of the design, and supply a user interface as part of the package.

Logiak has now been used to implement two ANGELIC designs. First we re-implemented the design for NIHL of [3]. Then as part of the exploration of extending ANGELIC to handle factors with magnitude we re-implemented the design described in [9] and [11] which added magnitudes to the well known US

Trade Secrets program CATO [4]. Since using Prolog to implement magnitudes posed significant problems in controlling unification and backtracking in [9], the ease of realisation of the CATO program in Logiak was especially encouraging.

The paper is structured as follows. Section 2 describes the ANGELIC design documentation, as used in the work with Logiak. Section 3 describes Logiak and its features, and its suitability for the realsiation of ANGELIC designs. Section 4 discusses the implementation of NIHL in Logiak. Section 5 describes the implementation of CATO using factors with magnitude in Logiak. Section 6 offers some discussion and concluding remarks.

2. ANGELIC

The ANGELIC methodology was described in [2]. It is designed to encapsulate the knowledge relating to a body of case law, to be used in factor based reasoning systems. Factor based reasoning is a well established approach to reasoning with legal cases, with its origins in HYPO [5] and CATO [4]. More recently the underlying precedential reasoning involved has been formalised [14]. For the history of the development of this style of reasoning see [8]. The methodology is based on traditional knowledge elicitation techniques, drawing the information from a variety documents (statutes, cases, commentaries, and training materials), under the guidance of a domain expert, who provides an introductory framework, advises on interpretation and validates the output. The most distinctive feature is the way in which the elicited knowledge is presented. This is in the form of a tree of factors, with the overall decision at the root, and with the base level factors, input by the user on the basis of the facts of the case, at the leaves. As one rises up the tree, factors become increasingly abstract and, as the root is approached, they become issues. This structure is very similar to the abstract factor hierarchy of CATO [4]. However, each node also has a set of *acceptance conditions* which state precisely how the children relate to the parent node. This feature enables the structure to be interpreted as an Abstract Dialectical Framework (ADF) [12]. Thus the design document provides both the advantages of a hierarchical structure, and a fine grained, domain relevant, partitioning of the knowledge base, while having the formal properties of the ADF. A fuller description of the stages can be found in [3], and examples of the design structures for the US Trade Secrets domain, the wild animals domain of [7] and the Automobile exception to the US Fourth Amendment can be found in [2].

Some adaptations were made to provide what was needed for Logiak, most notably the association of questions with each of the base level factors to support the provision of an interface. More changes were made to enable ANGELIC to accommodate factors with magnitude, as described in section 5.1. The most significant of these was the use of a 2-regular ADF [6], and the use of stereotypical patterns as the acceptance conditions, described in section 5.

3. Logiak

Logiak, produced by Things Prime GmbH, is a system with two main aspects.

- Firstly, it is a “no code” environment within which it is possible to create systems, including mobile systems, by configuration only.
- Secondly, as its name suggests, it is a system concerned to facilitate the representation of complex decision logic.

The design of Logiak has been influenced by its deployment in projects which use mobile technology to support often poorly-trained health workers in under-resourced settings to nevertheless follow best practice in diagnostic and treatment logic[1].

Diagnosis and treatment logic can be very complex logic indeed and, while the WHO’s recent Digital Health Guideline[2] affirms the use of decision support software to improve the quality of care provided, at the same time it remarks on “the importance of ensuring the validity of the underlying information, such as the algorithms and decision-logic”.

The challenge of ensuring correctness in the implementation of complex logic is one of the main reasons why we are focussed on having a “no code” approach. We think it essential that the logic implemented is made transparent to domain experts (whether they be doctors or lawyers), both for verification and for explanation of the outputs.

3.1. Declarative/Procedural

Implementing logic which can be executed is a challenge. Logic programming (e.g. with Prolog) is the ambitious effort to permit the use of predicate logic alone to be sufficient to achieve a complex executable system, but the exclusion of explicit procedural control proves in reality to be a real problem for projects and can lead to some compromised Prolog. Implementations based on ANGELIC exhibit this need to introduce procedural aspects into the Prolog both in [2] and, to an even greater extent, in [9]. Also, not all “logic” one wants to implement is declarative: sometimes one needs to describe procedures.

Logiak permits explicit representation of both procedural and declarative logic and clearly separates the two. In Logiak, one defines “processes” in two parts: “nodes” and “conditions”.

The “nodes” are sequential and each represents either an interaction with the user (e.g. asking the user a question and obtaining input) or a background action (e.g. updating a variable or updating the database).

A Process is therefore a sequence of such nodes executed one after the other. However, the execution can be affected by the specification of “preconditions” for nodes or groups of nodes (if a precondition is not true, the node is not executed). Such conditions are defined purely declaratively, either in terms of values

[1]Two examples of global health projects using Logiak: *Médicine sans Frontières* use Logiak to create and maintain their pediatric care decision support system eCARE-ped (see https://vimeo.com/msfch/review/301565776/2962c6151e), and it forms the basis of an emergency transport system in Tanzania and Lesotho (see https://www.vodafone.com/content/index/media/vodafone-groupreleases/2016/maternal-health-tanzania.html)

[2]https://www.who.int/reproductivehealth/publications/digital-interventions-health-system-strengthening/en/

of variables or responses from the user. Additionally, and importantly, one can define "meta-conditions" – i.e. conditions can be logical combinations of other conditions.

Experience has shown that the clean separation of the declarative from the procedural means that it is straightforward for domain experts to become fluent in specifying the fundamental logic of a Process.

3.2. Logiak and ANGELIC

Addressing the same problem (representing complex decision logic) from different directions, we observe that perhaps not entirely surprisingly the Logiak and Angelic representations have close structural similarities. These similarities, as we shall show, made it remarkably straightforward to use Logiak to activate ANGELIC designs into functioning systems.

4. Noise Induced Hearing Loss (NIHL)

Hearing loss induced by noise to which workers are subjected as part of their employment is widespread and it is possible for workers to make claims for compensation against negligent employers. Weightmans act for employers and their insurance companies by advising on whether claims should be settled or contested. The NIHL application was implemented in Logiak as a proof of concept of the compatibility of ANGELIC and Logiak, and to demonstrate the interface produced from Logiak.

Table 1. Selected nodes from NIHL design document

ID	Factor	Children	Conditions	Description
20	Breach of Duty	26 Employee told of Risks 27 Methods to reduce noise 28 Protection zone 29 Health surveillance 30 Risk assessment	REJECT IF Employee told of Risks AND Methods to reduce noise AND Protection zone AND Health Surveillance AND Risk assessment ACCEPT otherwise	The employer did not follow the code of practice is some respect.
28	Protection zone	Base Level	Q6 Yes	Employer provides methods to identify areas where noise level are high

4.1. Design

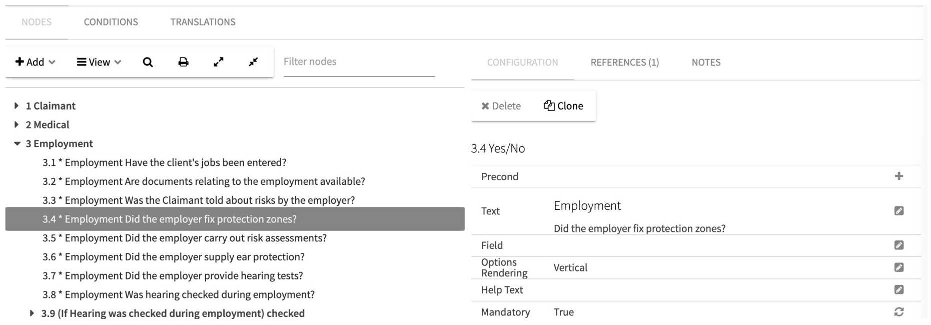
The design document used in the NIHL application was essentially the same as that produced in [3]. The only difference was that the base level factors were now associated with a question to be posed to the user. A set of questions and possible responses, taken from the check-list document used in the elicitation and the interface designed for [3] were supplied. This is so that the interface can also be generated from the document. The rows for the node *BreachOfDuty* and one of its base level children are shown in Table 1. Question 6, used to give a value to *Protection zone*, was *Did the employer fix protection zones? Yes/No.*

This design was then realised using Logiak as described in the next section.

4.2. Realisation

It is not an exaggeration to say that, the task of taking the ANGELIC specification of NIHL and using Logiak to create a functioning interactive system was largely a matter of (simply) transcribing the design document elements.

The first kind of transcription is to take the questions associated with base level factors and enter them into a Logiak Process, thus creating a user dialogue (see Figure 1).



1 Claimant
2 Medical
3 Employment
3.1 * Employment Have the client's jobs been entered?
3.2 * Employment Are documents relating to the employment available?
3.3 * Employment Was the Claimant told about risks by the employer?
3.4 * Employment Did the employer fix protection zones?
3.5 * Employment Did the employer carry out risk assessments?
3.6 * Employment Did the employer supply ear protection?
3.7 * Employment Did the employer provide hearing tests?
3.8 * Employment Was hearing checked during employment?
3.9 (If Hearing was checked during employment) checked

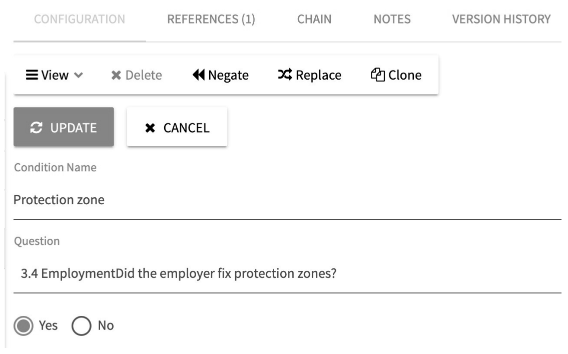
3.4 Yes/No

Precond	
Text	Employment Did the employer fix protection zones?
Field	
Options Rendering	Vertical
Help Text	
Mandatory	True

Figure 1. Questions within a Process

The second (and more interesting) "transcription" relates to the logic: one defines the conditions in Logiak, in a way which closely mirrors the acceptance conditions defined in the ANGELIC specification. In Logiak, one can define conditions of various types. The simplest are those defined on the basis of user responses to questions, and so correspond directly to ANGELIC "base level factors". For example, for the yes/no question "Did the employer fix protection zones?" in Figure 1, we define a condition named "Protection Zone" which is true if and only if the user responded affirmatively to said question (Figure 2).

On the basis of these conditions corresponding to ANGELIC "base level factors", in Logiak we can define "meta conditions" which are logical combinations of other conditions. For example, for the ANGELIC "Breach of Duty" conditions, we defined a Logiak meta-condition "No breach of duty" as shown in Figure 3, which is true if all base conditions relating to employer duties are satisfied and



Condition Name

Protection zone

Question

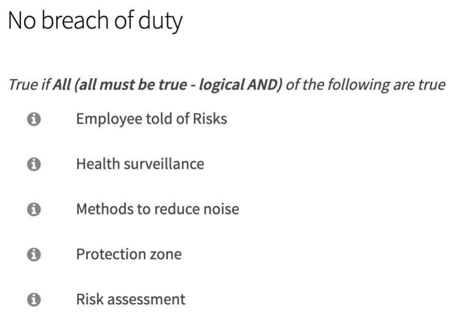
3.4 EmploymentDid the employer fix protection zones?

[✓ Yes] No

Figure 2. Defining a condition on the basis of user input

false otherwise. We then defined a meta-condition “Breach of Duty” which is true if the “No breach of duty” is false.

This indicates that the only aspect of implementing a system in Logiak based on ANGELIC which is not effectively transcribing the ANGELIC methodology output in a one-to-one manner, is in mapping the accept-reject logic of ANGELIC into declarative logic. ANGELIC makes use of defaults, for example, whereas in Logiak all conditions must be explicit. In practice, this poses little difficulty.



No breach of duty

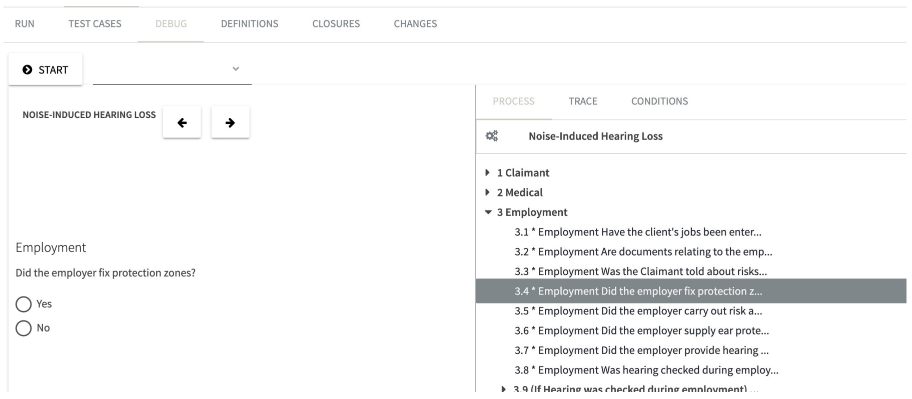
*True if **All (all must be true - logical AND)** of the following are true*

- Employee told of Risks
- Health surveillance
- Methods to reduce noise
- Protection zone
- Risk assessment

Figure 3. A ”meta-condition” definition in Logiak

After these two kinds of “transcription” from ANGELIC, one has defined an interactive process in Logiak which can be delivered either on the web or as a mobile app without any further programming. Users can respond to the questions defined and Logiak will compute the logic dictated by the conditions and feedback to the user can be tailored on the basis of the logical outcomes. It can also be combined with other processes and embedded into a data handling application which can retain data on cases for processing over time.

Within the Logiak environment, one can interact with a process defined to check and debug the logic as portrayed in Figure 4. The interface that will be seen by end users is shown in the left hand pane. If desired, the question shown to users can be accompanied by explanatory text and pictures.



NOISE-INDUCED HEARING LOSS

Employment

Did the employer fix protection zones?

◯ Yes
◯ No

Noise-Induced Hearing Loss

- 1 Claimant
- 2 Medical
- 3 Employment
  - 3.1 * Employment Have the client's jobs been enter...
  - 3.2 * Employment Are documents relating to the emp...
  - 3.3 * Employment Was the Claimant told about risks...
  - 3.4 * Employment Did the employer fix protection z...
  - 3.5 * Employment Did the employer carry out risk a...
  - 3.6 * Employment Did the employer supply ear prote...
  - 3.7 * Employment Did the employer provide hearing ...
  - 3.8 * Employment Was hearing checked during employ...
  - 3.9 (If Hearing was checked during employment) ...

Figure 4. Executing the Process in the debugger

5. CATO with Magnitudes

The CATO application was somewhat more ambitious than the NIHL application, in that it used factors with magnitude as well as Boolean factors. The need for some factors to have magnitude has become widely recognised in AI and Law [10], [16] and [15]. In particular the need for magnitudes in CATO was discussed in [6].

5.1. Design

In order to explore the suitability of ANGELIC to adapt to factors with magnitude, the Boolean design used in [2] was rewritten with some of the base level factors given magnitudes. This involved some changes to the original design, in order that the truth conditions could be written so as to accommodate non-Boolean factors. One change was to rewrite the original ADF of [2] as a 2-regular ADF (given in full in [1]) in which every parent node has exactly two children. This faciltates implementation by making the treatment of nodes more uniform. This design was implemented in Prolog [9], but the code was extremely procedural and rather laborious to construct because a fine grain level of control had to be imposed.

What this exercise did achieve, however, was the identification of a limited number of patterns. Building on [9], some twelve patterns were identified for the current exercise:

And: Node = Min(child1,child2)
Or: Node = Max(child1,child2)
Sum: Node = weight(val(child1)) * degree(child1)
+ weight(val(child2)) * degree(child2)
Difference: Node = weight(val(child1)) * degree(child1)
- weight(val(child2)) * degree(child2))
Exception1: If degree(child1) > threshold(val(child1)) then node = 1
else node = degree(child2).

Table 2. Abstract Factors in CATO application

Parent	Type	Value	Child 1	Child 2	Condition Pattern
Known	Boolean	LM	Limitations	KnownOutside	ThresholdException
IllegalMethods	Boolean	QM	Criminal	Dubious	Or

Table 3. Base Level Factors in CATO application

Factor	Type	Question	Pattern
AgreedNotToDisclose	Boolean	Did the defendant agree not to disclose?	QueryTheUser1
SecurityMeasures	Magnitude	On a scale of 0-10, how strong were the security measures taken by the plaintiff?	QueryTheUser3 (10)

Exception2: If degree(child1) > threshold(val(child1)) then node = 0 else node = degree(child2).
ThresholdException1: If degree(child1) > threshold(value(child1)) then node = 0 else if degree(child2) > threshold(value(child2)) then node =1 else node = 0,
ThresholdException2: If degree(child1) > threshold(value(child1)) then node = 1 else if degree(child2) > threshold(value(child2)) then node = 0 else node = 1.
ThresholdOr: If degree(child1) > threshold(value(child1)) then node = 1 else if degree(child2) > threshold(value(child2)) then node = 1 else node = 0.
QueryTheUser1: If *yes* Node = 1 else if *no* Node = 0
QueryTheUser2: If *yes* Node = 0 else if *no* Node = 1
QueryTheUser3(M): User inputs a number N. Node = N/M

Instead of acceptance conditions, each node is now associated with one of these twelve patterns, showing how the parent relates to its children. The base level factors are associated with a question and one of the *Query the User* patterns. Note that the patterns require the specification of *weights* and *thresholds*. Rather than specifying these for each individual factor, we specify the weights and thresholds for *values*, and so each factor needs to be associated with a value. We use the five values identified for CATO in [13], and associate them with factors as in [1]. The weights and thresholds can be set to reflect the relative importance of the values: the more important values are associated with higher weights to increase their importance in the weighted sum pattern, and more important values are associated with lower thresholds so that a lesser degree of satisfaction may still have an impact. In fact, we used equal weights and thresholds in this project. Effects of varying the weights and thresholds are discussed in [11].

Example nodes for abstract factors are shown in Table 2, and example base level factors are shown in Table 3.

5.2. Realisation

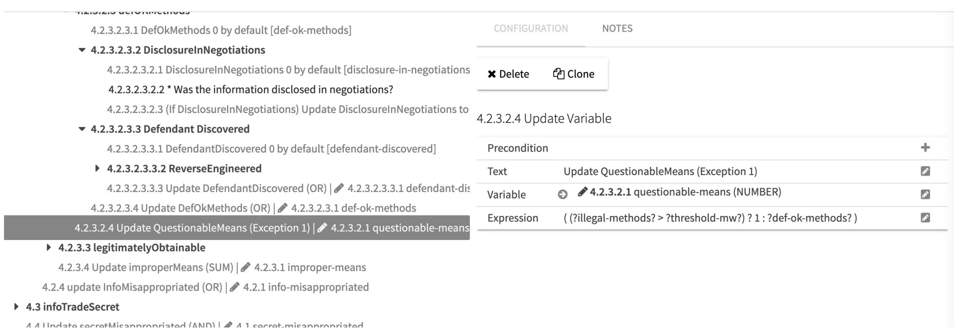
Within Logiak, a Process can contain not only interactions, such as the questions to the user as described above, but also actions. Actions are Process steps which happen in the background without user interaction and can include, for example, the creation and updating of variables.

Conditions can be defined on the values of such variables, just as they can be defined on user responses, so the implementation of reasoning with factors can also be achieved in the Logiak environment.

Actions which update numerical variables can use an expression language and the task of reflecting ANGELIC's use of factors became effectively the inclusion of variable update actions using expressions which implement the patterns described in the previous section.

Where ANGELIC has *Or: Node = Max(child1,child2)*, in Logiak expression language, we can update a variable using an expression of the form *(CONDITION ? VALUE-IF-TRUE : VALUE-IF-FALSE)*, where the condition compares values for child1 and child2, so *Or* is written *(child1 > child2 ? child1 : child2)*.

In illustration, Figure 5 shows a concrete example Action node with an expression which implements pattern "Exception 1" to update the value of a variable, *questionable-means* (a factor in CATO and a node in the ANGELIC design):



4.2.3.2.3.1 DefOkMethods 0 by default [def-ok-methods]
4.2.3.2.3.2 DisclosureInNegotiations
4.2.3.2.3.2.1 DisclosureInNegotiations 0 by default [disclosure-in-negotiations
4.2.3.2.3.2.2 * Was the information disclosed in negotiations?
4.2.3.2.3.2.3 (If DisclosureInNegotiations) Update DisclosureInNegotiations to
4.2.3.2.3.3 Defendant Discovered
4.2.3.2.3.3.1 DefendantDiscovered 0 by default [defendant-discovered]
4.2.3.2.3.3.2 ReverseEngineered
4.2.3.2.3.3.3 Update DefendantDiscovered (OR) | 4.2.3.2.3.3.1 defendant-dis
4.2.3.2.3.4 Update DefOkMethods (OR) | 4.2.3.2.3.1 def-ok-methods
4.2.3.2.4 Update QuestionableMeans (Exception 1) | 4.2.3.2.1 questionable-means
4.2.3.3 legitimatelyObtainable
4.2.3.4 Update improperMeans (SUM) | 4.2.3.1 improper-means
4.2.4 update InfoMisappropriated (OR) | 4.2.1 info-misappropriated
4.3 infoTradeSecret
4.4 Update secretMisappropriated (AND) | 4.1 secret-misappropriated

4.2.3.2.4 Update Variable

Precondition	
Text	Update QuestionableMeans (Exception 1)
Variable	4.2.3.2.1 questionable-means (NUMBER)
Expression	((?illegal-methods? > ?threshold-mw?) ? 1 : ?def-ok-methods?)

Figure 5. Variable update action implementing Exception1 pattern

Implementing the ANGELIC ADF with factors with magnitude therefore significantly increased the number of nodes in the Logiak Process and hence the complexity of the implementation. The 2-regular ADF, however, meant that although the result was a Process with many nodes, its production was in fact the quasi-mechanical application of patterns.

We should note that the implementation in Logiak could be made even simpler by a direct association of a magnitude with each Condition, where Conditions represent ANGELIC factors, allied with the implementation of the "patterns" above, not as explicitly constructed expressions but as system operators alongside the normal "OR" and "AND" etc. This would shift a lot of the complexity once again away from the Process to be hidden from the implementer, which would be once again more in line with the "no code" ethos of the system (since a Process with many variables and variable updates begins to be something challenging for non-programmers to manage).

6. Discussion and Concluding Remarks

Both implementations were evaluated against the applications produced in [3] and [9]. They were run using the same test data and produced fully correct results. The close structural correspondence between Logiak and ANGELIC greatly facilitated the verification of the implementation against the design. Moreover the discipline imposed by the implementation meant that any imperfections and unclarities could be detected and resolved. The CATO exercise threw up 15, mostly minor, queries, leading to a better design. Moreover, the immediate availability of a user interface meant that end users could be involved in evaluation. Weightmans provided positive feedback on the NIHL application

The ability to rapidly turn the design into a useable application greatly enhances the development process, by identifying problems at early stage so that the design can be refined, and by enabling end users and domain experts to participate in the process using the interface which is part of the Logiak package. Further, implementation in Logiak means that it is unnecessary to develop a separate user interface, which required a substantial additional effort for NIHL [3]. The ability to provide a straightforward way of implementing the ANGELIC designs is an important addition to the methodology, greatly increasing its practical usability.

References

[1] L Al-Abdulkarim, K Atkinson, and T Bench-Capon. Factors, issues and values: Revisiting reasoning with cases. In *Proceedings of the 15th ICAIL*, pages 3–12. ACM, 2015.

[2] L Al-Abdulkarim, K Atkinson, and T Bench-Capon. A methodology for designing systems to reason with legal cases using ADFs. *AI and Law*, 24(1):1–49, 2016.

[3] L Al-Abdulkarim, K Atkinson, T Bench-Capon, S Whittle, R Williams, and C Wolfenden. Noise induced hearing loss: Building an application using the angelic methodology. *Argument & Computation*, 10(1):5–22, 2019.

[4] V. Aleven. *Teaching case-based argumentation through a model and examples*. PhD thesis, University of Pittsburgh, 1997.

[5] K Ashley. *Modeling legal arguments: Reasoning with cases and hypotheticals*. MIT press, Cambridge, Mass., 1990.

[6] K Atkinson and T Bench-Capon. Dimensions and values for reasoning with legal cases. *Tech Report ULCS 17-004, Department of Computer Science, U of Liverpool*, 2017.

[7] Katie Atkinson. Introduction to special issue on modelling Popov v. Hayashi. *Artificial Intelligence and Law*, 20(1):1–14, 2012.

[8] T Bench-Capon. HYPO's legacy: introduction to the virtual special issue. *Artificial Intelligence and Law*, 25(2):205–250, 2017.

[9] T Bench-Capon and K Atkinson. Implementing factors with magnitude. In *Proceedings of COMMA 2018*, pages 449–450, 2018.

[10] T Bench-Capon and E Rissland. Back to the future: Dimensions revisited. In *Proceedings of JURIX 2001*, pages 41–52. IOS Press, 2001.

[11] Trevor Bench-Capon and Katie Atkinson. Lessons from implementing factors with magnitude. In *Proceedings of JURIX 2018*, pages 11–20, 2018.

[12] G Brewka, S Ellmauthaler, H Strass, J Wallner, and P Woltran. Abstract dialectical frameworks revisited. In *Proceedings of the Twenty-Third IJCAI*, pages 803–809. AAAI Press, 2013.

[13] A Chorley and T Bench-Capon. An empirical investigation of reasoning with legal cases through theory construction and application. *AI and Law*, 13(3):323–371, 2005.

[14] J Horty and T Bench-Capon. A factor-based definition of precedential constraint. *AI and Law*, 20(2):181–214, 2012.

[15] John Horty. Reasoning with dimensions and magnitudes. *Artificial Intelligence and Law*, pages 1–37, 2019.

[16] Adam Rigoni. Representing dimensions within the reason model of precedent. *Artificial Intelligence and Law*, 26(1):1–22, 2018.