Realising ANGELIC Designs Using Logiak

Katie Atkinson[a], Trevor Bench-Capon[a], Tom Routen[b], Alejandro Sánchez[b]
Stuart Whittle[c], Rob Williams[c] and Catriona Wolfenden[c]
a Department of Computer Science, The University of Liverpool, UK:
b Things Prime GmbH, Basel, Switzerland. routen@mangologic.com:
c Weightmans LLP, Liverpool, UK

Abstract. ANGELIC is a methodology for encapsulating knowledge of a body of case law. Logiak is a system intended to support the development of logic programs by domain experts, and provides an excellent environment for the rapid realisation of ANGELIC designs. We report our use of Logiak to realise ANGELIC designs, using both Boolean factors and factors with magnitude.

1. Introduction

The ANGELIC methodology for designing systems intended to encapsulate case law was described in [1]. 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 [2]. Realising the design required considerable effort in [2] and a custom built interface had to be produced from scratch. To address these problems we explored the use of a target implementation platform, Logiak, to enable rapid and convenient realisation of the design, and to supply a user interface as part of the package.

Logiak has been used to implement two ANGELIC designs. First we re-implemented the design for NIHL of [2]. Then as part of the exploration of extending ANGELIC to handle factors with magnitude we re-implemented the design described in [6] which added magnitudes to the well known US Trade Secrets program CATO [3]. A longer version of this paper [4] which supplies additional details can be found at https://cgi.csc.liv.ac.uk/research/techreports/.

2. ANGELIC and Logiak

The ANGELIC methodology described in [1] is designed to encapsulate case law knowledge to be used in factor based reasoning systems [5]. The methodology is based on traditional knowledge elicitation techniques, drawing the information from a variety documents under the guidance of a domain expert. The knowledge

is presented in a form similar to the abstract factor hierarchy of CATO [3]. However, each node also has a set of *acceptance conditions* which state precisely how children relate to their parent node and enable the structure to be interpreted as an Abstract Dialectical Framework (ADF) [7]. 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 of the methodology can be found in [2], and several examples of design structures in [1]. 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. The most significant of these was the use of a limited number of design patterns as the acceptance conditions.

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. Users include *Médicines sans Frontières*. Diagnosis and treatment logic can be very complex logic indeed and, while the WHO's Digital Health Guideline (www.who.int/reproductivehealth/publications/digital-interventions-health-system-strengthening/en/) affirms the use of decision support software to improve the quality of care, it remarks on "the importance of ensuring the validity of the underlying information, such as the algorithms and decision-logic".

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. obtaining input) or a background action (e.g. updating a variable). 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 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. 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 com-

pensation 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 in some respect.
28	Protection zone	Base Level	Q6 Yes	Employer provides methods to identify areas where noise level are high

3.1. Design

The design document used for the NIHL application was essentially the same as that produced in [2]. 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 [2], were supplied 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.

3.2. Realisation

Using Logiak to create a functioning interactive system from the ANGELIC specification of NIHLwas 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, to create a user dialogue (shown as Figure 1 of [4]).

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?" we define a condition named "Protection Zone" which is true if and only if the user responded affirmatively to said question (Figure 2 of [4]).

Using these conditions (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" (shown in Figure 3 of [4]), which is true if all base conditions relating to employer duties are satisfied and 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.

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 and Logiak will compute the logic dictated by the conditions. Within the Logiak environment, one can interact with a process defined to check and debug the logic as portrayed in Figure 1. 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.

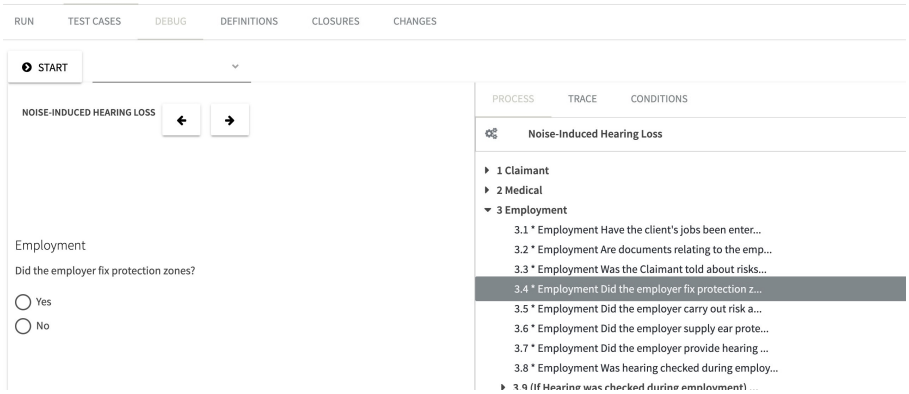

Figure 1. Executing the Process in the debugger

4. CATO with Magnitudes

The CATO application 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: in particular the need for magnitudes in CATO was discussed in [6].

Table 2. Abstract Factors in CATO. LM is Legitimate Means and QM is Questionable Means

Parent	Type	Value	Child 1	Child 2	Condition Pattern
Known	Boolean	LM	Limitations	KnownOutside	ThresholdException2
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)

4.1. Design

To adapt to factors with magnitude, the Boolean design used in [1] was rewritten with some of the base level factors given magnitudes. This involved some changes to the original design, in order that the acceptance conditions could be written so as to accommodate non-Boolean factors. One change was to rewrite the original ADF of [1] as a 2-regular ADF (shown in full in [6]) in which every parent node has exactly two children. This facilitates implementation by making the treatment of nodes more uniform. This design was implemented in Prolog [6], but the code was extremely procedural and rather laborious to construct because a fine grained level of control had to be imposed. What this exercise did achieve, however, was the identification of a limited number of patterns for acceptance conditions. Building on [6], twelve patterns were identified for the current exercise. The twelve patterns were *And*, *Or*, *Weighted Sum*, *Weighted Difference*, 2 kinds of *exception*, 3 uses of *thresholds* and 3 varieties of *Query the User*. For details of the patterns see [4].

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*. These were specified for *values* rather than individual factors, and so each factor needs to be associated with a value. We used the five values identified for CATO in [8]. The weights and thresholds can be set to reflect the relative importance of the values but we used equal weights and thresholds. Effects of varying the weights and thresholds are discussed in [6]. Example nodes for abstract factors are shown in Table 2, and example base level factors are shown in Table 3.

4.2. 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. Actions which update numeric 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 above, making it a quasi-mechanical process. The implementation in Logiak could be simplified by a direct association of a magnitude with each Condition representing an ANGELIC factor, and implementing the "patterns" above, not as explicitly constructed expressions but as system operators. This would also hide the detailed expressions from the implementer, which would be more in line with the "no code" ethos of the system.

5. Discussion and Concluding Remarks

Both implementations were evaluated against the applications described in [2] and [6]. 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 [2]. Providing a straightforward way of implementing 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. A methodology for designing systems to reason with legal cases using ADFs. *AI and Law*, 24(1):1–49, 2016.

[2] 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.

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

[4] K Atkinson, T Bench-Capon, Routen T, Alejandro Sánchez, Stuart Whittle, Rob Williams, and Catriona Wolfenden. Implementing ANGELIC designs using logiak. Technical Report ULCS-19-002, University of Liverpool, 2019.

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

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

[7] G Brewka and S Woltran. Abstract Dialectical Frameworks. In *Twelfth International Conference on the Principles of Knowledge Representation and Reasoning*, 2010.

[8] 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.