On Building a Visualisation Tool for Access Control Policies

Charles Morisset[1](✉) and David Sanchez[2]

[1] Newcastle University, Newcastle Upon Tyne, UK
charles.morisset@ncl.ac.uk

[2] Northumbria University, Newcastle Upon Tyne, UK
david.sanchez@northumbria.ac.uk

Abstract. An access control policy usually consists of a structured set of rules describing when an access to a resource should be permitted or denied, based on the attributes of the different entities involved in the access request. A policy containing a large number of rules and attributes can be hard to navigate, making policy editing and fixing a complex task. In some contexts, visualisation techniques are known to be helpful when dealing with similar amounts of complexity; however, finding a useful visual representation is a long process that requires observation, supposition, testing and refinement. In this paper, we report on the design process for a visualisation tool for access control policies, which led to the tool VisABAC. We first present a comprehensive survey of the existing literature, followed by the description of the participatory design for VisABAC. We then describe VisABAC itself, a tool that implements *Logic Circle Packing* to pursue the reduction of cognitive load on Access Control Policies. VisABAC is a web-page component, developed in Javascript using the D3.js library, and easily usable without any particular setup. Finally, we present a testing methodology that we developed to prove usability by conducting a controlled experiment with 32 volunteers; we asked them to change some attribute values in order to obtain a given decision for a policy and measured the time taken by participant to conduct these tasks (the faster, the better). We obtained a small to medium effect size ($d = 0.44$) that indicates that VisABAC is a promising tool for authoring and editing access control policies.

Keywords: Visualisation · Attribute-based Access Control · User study · Circle Packing

1 Introduction

An *access control policy* can be seen as a compendium of authorisations that regulate the use of a particular set of resources. They are defined by *security administrators* and are processed by a trusted software module called *access control mechanism* or *reference monitor* [8].

The first access control model is usually considered to be the *Access Matrix* [21], in which a head-rows indicate subjects, head-columns represent objects and the interception-cells, the access rights granted. This approach could be inconvenient for systems requiring a large number of subjects and objects, and may lead to policy misconfigurations [6]. As a consequence, alternative access control models have been introduced[1] over the years that not only provide more convenient methods for designing policies in specific contexts but also aim for more expressivity. In that quest, *General policy languages* have subsequently been created, including, but not limited to, *ExPDT* [43], *EPAL* [2] and the standard XACML (eXtensible Access Control Markup Language) [45]. The latest version, XACML 3.0, was released in 2013, and standardizes *Attribute-based Access Control*, within which an *access request* can be seen as a set of attribute values, an *access rule* as a decision (e.g., permit or deny) returned when a boolean expression (i.e., target and/or condition) holds for a request, and an *access policy* as combining the decisions returned by a collection of rules using a composition operator (e.g., deny-overrides or permit-overrides). Although XACML is a very general and powerful framework, its underlying format is XML, which makes XACML policies machine readable, but arguably harder to author and edit by hand.

The need for including human factors —which involve human-software interaction [22]— in security is recognised as an important problem; in the UK, for instance, 50% of the worst 2015 breaches were caused by "inadvertent human error" (up from 31% in 2014) [36] and there has been an increasing effort on *usable security* (see, e.g., [1,17,20,48]). In fact, human cognitive capacity has been overflowed to such extend by the need of regulatory mandates [4] that typical Security Administrators cope with such entanglement by obviating irrelevant data, causing inadvertently security risks in the process [49]. Recent privacy breaches along with experiments, such as Trudeau et al. [48] corroborates this, showing that users (including experienced policy engineers) easily oversee details. There is therefore a clear case to build tools helping security administrators author and edit access control policies.

Reducing complexity is an essential stage in any kind of analysis and it is perfectly possible to simplify a system without loosing essential functional properties. *Information visualisation* [9] comprises techniques that allow humans to understand and manipulate huge quantities of abstract data by simplification and it is being actively investigated by security researchers [7,47,49]. Languages such as `Mir6` [15] have demonstrated that it is even possible to specify security visually, albeit with very limited complexity. In particular, visualisation techniques have been proposed in the context of access control [15,42], including the tools ALFA[2] (Axiomatics Language for Authorization), which proposes a much simplified textual syntax for describing XACML policies, or VisPE [29], which proposes a Sratch-based interface. However, these approaches tend to enhance

[1] See for instance [3] for an account on the variety of access control models introduced over the past decades.

[2] https://www.axiomatics.com/pure-xacml.html.

the textual representation of the policy, rather than offer a visualisation of the evaluation of a policy.

In this paper, we fully report on the design process we followed to design VisABAC [26], which provides an interface for evaluating access control policies represented by Circle Packing drawing technique[3]. Whereas our previous work [26] focuses on the result of a usability study we conducted to validate VisABAC, this paper focuses on the different steps of the design process as well as the description of the tool itself. More specifically, the contributions of this paper are:

– A comprehensive description of the design process for VisABAC was created. A considerable amount of time was invested into exploring concepts and ideas that have been detailed in the background section (Particularly in Sect. 2.2); we detailed early prototypes in Sect. 3.1.
– A comprehensive description of VisABAC and its inner workings. VisABAC is a client-side browser application that given an attribute-based access control policy, provides a textual representation of that policy (inspired by XACML 3.0 and ALFA), a graphical visualisation using the Circle Packing method, and an interface allowing a policy designer to change policy and attribute values. VisABAC is, to the best of our knowledge, the first visualisation tool to support the XACML 3.0 extended decision set, which includes multiple indeterminate decisions (indicating missing information).
– An explanation of the methodology we designed to measure usability of a visualisation tool, as well as the report of a controlled experiment with 32 participants, which showed that, compared to the controlled group, the tested group was, in average, faster to answer the questions (with an effect size of $d = 0.44$ over the monitored questions), and more likely to interact with the tool (subjective preferences measured at the end of the test showed that 76.47% of participants who tested the visualisation tool manifested they felt more confident operating the policy.)

These contributions can be particularly helpful to those intending to design a visualisation tool for access control policies, as we highlight the key problems we have encountered in the design process. In particular, to the best of our knowledge, there is no standard benchmark for evaluating the efficiency and usability of policy authoring/editing tool, and we believe the results of the controlled experiments could pave the way towards establishing such a benchmark.

The rest of this paper is structured as follows: we first introduce in Sect. 2 the background on Attribute-Based Access Control and related work on access control visualisation. We then present VisABAC in Sect. 3, how it was developed and inner workings essentials; the experiment in Sect. 4; results are discussed in Sect. 5 and conclusions in Sect. 6.

[3] VisABAC is open-source and available at https://gitlab.com/morisset/visabac.

2 Background and Related Work

2.1 ABAC

As briefly described in the Introduction, ABAC consists in considering an access request as a set of attribute values. To illustrate our approach, let us consider a health-care policy, regulating the access to a medical record, where, informally speaking, access is permitted when there is no explicit disagreement from the patient and when either the hospital or the concerned surgeon agrees for the access, and access is denied otherwise.

We present here a simplified version of ABAC, aligned with the current version of VisABAC, and we leave for future work the implementation of more complex ABAC languages, such as PTaCL [11,12]. This simplified version is nevertheless expressive enough to model missing information, which is a key aspect of XACML 3.0 and PTaCL. In a nutshell, we consider here five key concepts:

– An *atomic target* consists of an attribute name and an attribute value;
– An *access request* provides a valuation of atomic targets to a 3-valued logic;
– A *composite target* is a logical composition of atomic targets;
– An *access rule* consists of an access decision and a composite target;
– An *access policy* composes rules and policies using a composition operator.

Intuitively speaking, we can define the policy described above using the following syntax (we provide a formal definition below):

```
R1: Deny if PATIENT_disagrees
R2: Permit if OR(HOSPITAL_agrees, SURGEON_agrees)
P: DOV(R1,R2)
```

where `PATIENT_disagrees`, `HOSPITAL_agrees` and `SURGEON_agrees` are atomic targets, `OR(HOSPITAL_agrees, SURGEON_agrees)` is a composite target, `Permit` and `Deny` are access decisions, `R1` and `R2` are access rules, `DOV` is the deny-overrides composition operator, and `P` is an access policy.

More formally, we consider a set of attribute names $\mathcal{A}$, a set of attribute values $\mathcal{V}$, and a set of atomic targets $\mathcal{T} \subseteq \mathcal{A} \times \mathcal{V}$. In the example above, we have $\mathcal{A} = \{\texttt{PATIENT}, \texttt{HOSPITAL}, \texttt{SURGEON}\}$, $\mathcal{V} = \{\texttt{agrees}, \texttt{disagrees}\}$, and $\mathcal{T} = \{(\texttt{PATIENT, disagrees}), (\texttt{HOSPITAL, agrees}), (\texttt{SURGEON, agrees})\}$ (we use the underscore notation in the textual representation to limit the number of parentheses). It is worth noting that we do not associate each attribute with each value. In practice, this can be quite significant, for instance with the encoding of the patient consent: in this example, we model an explicit disagreement instead of an explicit agreement.

A request is then defined as a function $q : \mathcal{T} \rightarrow \{1, 0, \bot\}$, such that, given an atomic target $t = (a, v)$, $q(t) = 1$ indicates that a has the value v in q, $q(t) = 0$ indicates that a does not have the value v in q, and $q(t) = \bot$ indicates that we do not know whether a has the value v in q or not. Here, we interpret 1, 0, and $\bot$ as the XACML elements Match, NoMatch and Indeterminate, respectively.

A composite target is defined as a proposition of atomic targets. Since, in the controlled experiment presented in Sect. 4, we targeted participants with no specific knowledge of access control, we only considered the conjunction ($\wedge$) and disjunction ($\vee$) operators, corresponding to the XACML AllOf and AnyOf elements, respectively. We leave the study of more complex logical operators for future work. We use a strong Kleene interpretation for the logical operators, following the PTaCL and XACML semantics: given a request q, and two targets t_1 and t_2, the target $t = t_1 \wedge t_2$ evaluates to 1 if both t_1 and t_2 evaluates to 1, to 0 if either t_1 or t_2 evaluates to 0, or to $\perp$ otherwise. Similarly, the target $t = t_1 \vee t_2$ evaluates to 1 if either t_1 or t_2 evaluates to 1, to 0 if both t_1 and t_2 evaluates to 0, or to $\perp$ otherwise.

An access rule is defined as a tuple (d, t), where d is a decision (either Permit or Deny) and t is a target. Given a request q, a rule (d, t) evaluates to d if t evaluates to 1, to NA (Not-Applicable) if t evaluates to 0, to Indet(P)[4] if $d =$ Permit and t evaluates to $\perp$, or to Indet(D) if $d =$ Deny and t evaluates to $\perp$.

Table 1. Evaluation of the healthcare policy example on some selected values for each atomic target, where each row corresponds to a different access request [26].

Targets				Rules		Policy
t_1	t_2	t_3	$t_2 \vee t_3$	r_1	r_2	p
1	1	1	1	Deny	Permit	Deny
0	1	1	1	NA	Permit	Permit
0	0	0	0	NA	NA	NA
0	$\perp$	0	$\perp$	NA	Indet(P)	Indet(P)
$\perp$	1	1	1	Indet(D)	Permit	Indet(PD)
$\perp$	0	0	0	Indet(D)	NA	Indet(D)

An access policy is a collection of rules, composed together with a composition operator. We implemented in VisABAC the six main XACML operators: permit-overrides (POV), deny-overrides (DOV), permit-unless-deny (PUD), deny-unless-permit (DUP), first-applicable (FA), only-one-applicable (OOA). We refer to the main documentation of XACML or for instance to [27] for the full definitions of these operators[5].

The example policy given above can be formally defined as follows: let $t_1 =$ (PATIENT,disagrees), $t_2 =$ (HOSPITAL,agrees) and $t_3 =$ (SURGEON,agrees) be atomic targets, $r_1 = (\mathsf{Deny}, t_1)$ and $r_2 = (\mathsf{Permit}, t_2 \vee t_3)$ be access rules, and $p = \mathtt{DOV}(r_1, r_2)$ the access policy. The evaluations of these elements are presented

[4] For the sake of compactness, we abbreviate the XACML Indeterminate extended decisions to Indet.

[5] Also available with VisABAC documentation: http://homepages.cs.ncl.ac.uk/charles.morisset/visabac/visualiser/resources/pages/help.html.

in Table 1. It is worth observing that this simple policy can in practice evaluate to every possible XACML decision, depending on the values of the atomic targets.

2.2 Visualisation for Access Control

As previously stated, cognitive overload is a major issue in access control policy design, deployment and maintenance and visualisation techniques could ease those processes; unfortunately, not all visualisation mechanisms are helpful since many of them grow too large for human cognition[6]. There exist a rich literature for visualisation in security, however, few approaches deal with Attribute-based Access Control, and these approaches tend to work on the structure of the policy itself, such as VisPE [29], rather than on policy evaluation.

In the following subsections we summarise different visualisation techniques, some of them actively applied into access control, that were considered in the process of building VisABAC.

Euler Diagrams [13] visually represent containment, intersection and exclusion using closed curves. They are largely used in math to represent set operations and deductive reasoning [41,44] and were the first kind of diagrams considered as VisABAC framework. Security lends naturally to this kind of visualisation since policies can be represented in terms of relationship sets and they have proved [38] to effectively visualise thousands of elements if the set intersection are simple; however, the method becomes almost unreadable when a low count of elements have complex relationships among them. Euler diagrams prove to be very inspiring in the prototype designed but they were not implemented since they could be particularly difficult to draw automatically [46].

Grids are matrices with policies along rows and resources as columns; results of the evaluation of access to resources are placed in intersections. For example, [37] propose the use of multi-level grids to visualise results of multiple types of access control policy analysis and authoring. This approach is very simple to implement yet very powerful; however it does not take advantage of many visualisation concepts and it is very space consuming.

Graphs are used to represent access policies. [25] explores them visually in operational situations with its RubaViz prototype; however, its main use has been as memory structures. [18], for instance, uses Multi-Terminal Binary Decision Diagrams (MTBDDs), as a way to model XACML policies in Margrave (a proposed software tool developed in Scheme). Even though it heavily uses

[6] As a side note, the abstractions and simplifications commonly used in visual techniques designed for humans, can also be useful to computers, presenting even formal proof of the correctness and normalisation of policies. For example, in [35] Graph theory is used to validate policies and in [30] decision diagrams are used to accelerate XACML speed evaluation; none of them show any visuals to users.

graphs, no visual representation is derived from its internal structure since even the memory arrangement of a very simple policy can generate a confusing graph for humans. As a consequence, a standard Graph visualisation was discarded in the early stages of the process, even though there are many interesting tools such as Gephi [5] which are worth to be considered in future research.

Shortcomings of the Graph approach can also be appreciated in PRISM (PRIvacy-Aware Secure Monitoring), a software tool that proposes an architecture to mediate between information sources and entities on a network presented by [28]. Access Policies controlled with this interesting visual editor provides user-friendly administration of complex X.509 certificates[7] by users with no particular expertise [28]. The interface lays out many instruments to interact with the graph, being possible zoom-in/out, rotate and navigate. As a drawback this kind of representation can become unreadable as the number of policies increases and the user can easily get lost inside the graphical representation. PRISM tries to minimise this by including a birds-eye view.

Trees are being timidly studied as a way to visually find conflicts inside access policies; this seems surprising since trees are used to create XACML policies itself and it is the preferred method for explaining XACML policies in the OASIS specification [39]. [42] explore this approach for very light graphs in its XACML Viz prototype. [35] uses trees (Matching tree and Combining Tree) to optimise the evaluation of applicable rules in an access policy engine called XEngine. This tool is not aimed at visualisation but uses visual concepts and matches internal structures directly to trees representations. Illustrations were used as inspirations for the prototype.

Semantic Substrates [9] uses spatial representation to group common attributes by regions. [33,34] propose a visualisation toolkit called "Policy Visualisation Framework (PVF)" which extends XACML to support RBAC. It aims at providing a clearer representation than a conventional role-permission tree graph, and it seems particularly useful when combining different policies. This visualisation technique mimics three electronic breadboards that represents user, role and permission. Nodes inside each breadboard are drawn as circles, squares and triangles; they are interconnected by red, green and blue lines (wires) which assign user-role, role-permission and role-mapping relationship respectively. Hierarchy is achieved by arrows in the relationships [33,34]. This technique has been successful when dealing with a relatively small number of policies but it has been insufficient with heavily dense policy graphs [51]. As a consequence, [51] propose complementing it with another techniques such as adjacency matrices.

[7] [28] indicates that future works is necessary in order to make PRISM a general purpose access control administration tool capable to support alternatives representations such as XACML.

Adjacency Matrices are widely used in graph visualisation because they allow a clear understanding of dense relationship structures. However, according [51] do not favoured them when dealing with hierarchical security relationships. As a consequence, [51] use them as a complementary representation to Semantic Substrate when visualising compliance of security policies in SELinux. Adjacency Matrices were discarded as a technique since they are not expressive enough.

Treemaps [16] visualise hierarchical tree structures using a root rectangle that contains all nodes of a given tree. Each subsequent level of the tree structure divides the above square according to a particular attribute of a node, such as size. [34] proposes treemaps to complement Semantic Substrates instead of adjacency matrices to form macro and micro vision respectively. It aims at the analysis of access control polices of RBAC model when multi-domain information is exchanged. Treemaps offer a perfect match between access policies and efficient space utilisation. They are pleasant to the eye and can provide interactivity. They were proposed for the prototype and survived along the first stages of implementation; unfortunately, they became difficult to understand as the number of policies increased and were finally discarded after pilot testing. Pictures of animated treemaps can be seen in Figs. 2 and 3.

Circle Packing [50] is very similar in concept to Treemaps, as it was inspired by them. As a marked difference, it uses circles instead of rectangles which give them a lower space efficiency ratio; however, they express more clearly the hierarchy they represent. Figure 1 shows a three-level Circle Packing diagram.

Even though Circle Packing may seem at first glance as Euler diagrams (or a type of Euler diagram like Venn), they are different in concept as well as in properties. For example, Circle Packing do not comply with many of the mathematical Euler characteristics, such as the presence of unique labels or crossing policies (Circle Packs do not intersect lines, while Euler diagrams do) [46]. However, for the purpose of VisABAC they provide the understandability of Euler representation with the ease of use and programmability of Treemaps. Additionally, Circle Packing provides clear containment —as Euler diagrams— but are space efficient. This is a huge benefit when comparing them with Trees, Grids and Graphs. [50] have shown with a file visualisation tool (FVT) that it is possible to handle efficiently thousands of nodes with this method. However, to the

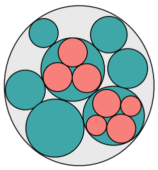


Fig. 1. Simple Circle Packing Diagram [50]. Level 0 is painted light grey. Level 1 is painted green. Level 3 is painted red. (Color figure online)

best of our knowledge, Circle Packing has never been used in the context of access control priory to the VisABAC [26] implementation.

3 VisABAC

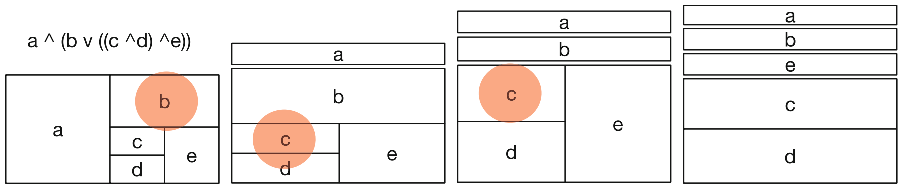
In this section, we first explain the process with which we have designed VisABAC, after which we describe the tool itself[8].

3.1 Creating VisABAC

In general, visualisation is not only a set of techniques but also a process [24] therefore, in order to achieve a successful representation, it is important to work closely with users affected by the shortcomings of traditional analysis. Hence, we work closely with 5 members of our research group using a *participatory design* [40]. That expertise targeted essential usability aspects and the feedback acquired (*heuristic approach* [40]) was complemented by *heuristic evaluation* and informal/formal evaluation by recruited participants.

VisABAC was developed using rapid-prototyping methodology [19]. The process involved three stages: throw-away, evolution and refinement.

The *Throw-away stage* involved the creation of over 70 prototypes with no functionality using presentation software to explore almost all ideas explained in the Sect. 2.2. Some approaches, such as: graphs, hierarchical graphs, hypergraphs, Euler diagrams, and binary decision diagrams (BDD), have already been identified as too complex to implement, visualise or unsuitable to be of any practical use [14,15,18,25]. Some candidates, on the other hand, were particularly promising, including trees and treemaps, which have been applied previously to security visualisation. Figure 2 shows an early prototype. In this stage, as well as the next one, we used simplified access control policies expressed as logical boolean algebra.



(a) Initial Treemap (b) Collapsed a (c) Collapsed b (d) Collapsed e

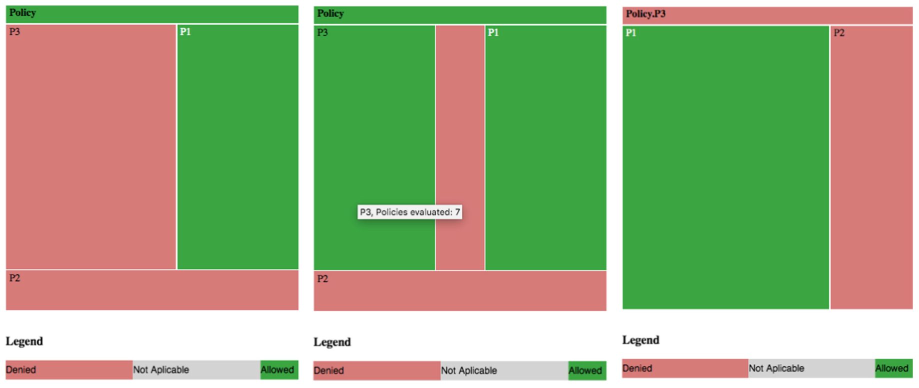
Fig. 2. Early prototype: (a) shows the initial stage with a logic equation representing an access control policy. Tapping on (a) advances to (b) and successively.

[8] VisABAC is available for demonstration at http://homepages.cs.ncl.ac.uk/charles.morisset/visabac.

The *Evolutionary stage* was started once some ideas were identified as possibly useful. Under this stage many limited functional prototype were created over a quick iteration process. These high-fidelity prototypes were developed on Javascript using the D3 library, coded using NetBeans 8.1 and displayed on a web browser (primarily Mozilla Firefox 47). These prototypes did not evaluate a full ABAC policy and instead used a simplified access control policy handcrafted in JSON. During this stage two very interesting prototypes emerged based on treemaps and trees. However, some limitations were found during the participatory process, even after trying to refine them as *zoomable treemaps* (Fig. 3) and *collapsible trees* (Fig. 4). The most relevant limitations were:

– Users easily forgot the evaluation result of a particular policy they were inspecting and had to waste time by going back to a previous level.
– The relationship between screen state utilisation and navigability was highlighted as very important by participants. Screen utilisation for collapsible trees was very low (more than 50% is background)[9] and caused excessive panning when dealing with large policies; on the other hand, zoomable treemaps proposed a full screen state utilisation but users got lost inside the policy quickly.

A tradeoff between efficiency and usability was found in *circle packing*, a visualisation technique criticised [50] for not being as space efficient as treemaps but praised for providing a better hierarchy illusion than those obtained by, for



(a) Treemap Level Policy (b) Treemap showing label (c) Treemap Level Policy.P3

Fig. 3. Zoommable Treemap prototype showing a simplified Access Control Policy (Policy) composed of 3 sub-policies (P1, P2 and P3). (a) shows the initial Treemap level O. (b) Hovering over P3 rectangle policy, the label reveals information about was performed the evaluation for this policy (How the colour was obtained). (c) tapping on P3 shows the immediate interior P3 level (Policy.P3).

[9] A prototype version of VisABAC with collapsible trees is available alongside the main tool, illustrating the poor screen utilisation.

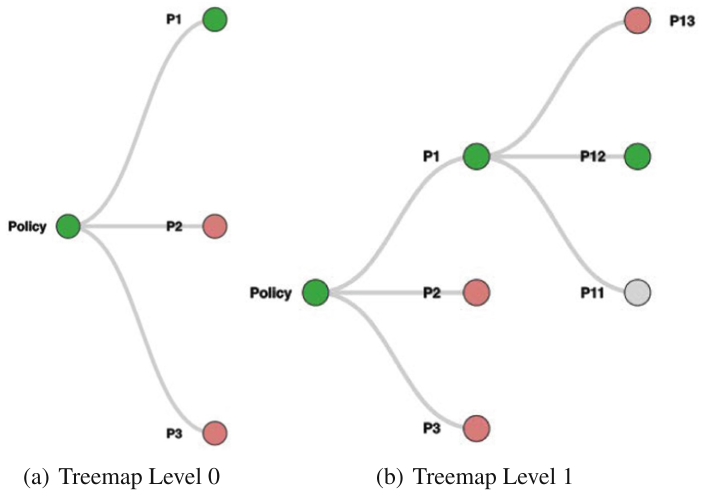


Fig. 4. Collapsible Tree prototype showing a simplified access control policy (Policy) composed of 3 sub-policies (P1, P2 and P3) (a) shows Policy with first level displayed (P1, P2 and P3). (b) shows P1 policy sublevel with their corresponding sub-sub-policies (P11, P12 and P13).

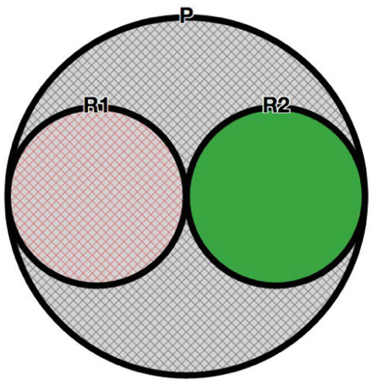
example, treemap representations. A late prototype of this stage was embedded into a FileMaker Pro application to combine an early version of the testing module with a database.

The *refinement stage* was started once the feasibility of the app was determined as well as a testing procedure could be applied. In this stage full access control policies could be edited and evaluated. Also, the FileMaker testing module was superseded by a Javascript one, making the new application completely web based. This final prototype became VisABAC and will be described in Sect. 3.2.

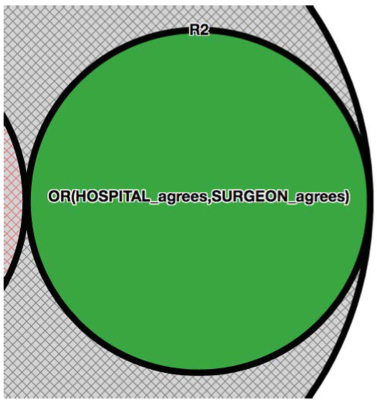
3.2 VisABAC Interface

The VisABAC interface is designed as a web page component and, as such, runs on any web browser. The interface consists of four main components, which we now detail, using the visualisation of the policy described in Sect. 2.1 as an example (Fig. 5).

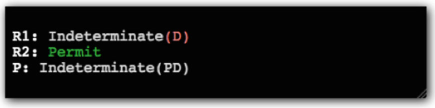
The *Policy component* (Fig. 5(d)) is a textual box, directly editable from the browser, which contains the definition of the policy following the syntax described in Sect. 2.1. This definition can either be typed in, loaded from a set of existing samples, or loaded from a file. These rules are automatically parsed into JavaScript Object Notation (JSON), where the text of each rule is identified by its name. For instance, the policy described in Sect. 2.1 would correspond to the object:



(a) Visual Decision Diagram

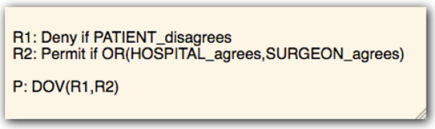


(b) Zoomed-in Diagram



```
R1: Indeterminate(D)
R2: Permit
P: Indeterminate(PD)
```

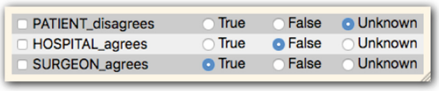
(c) Decision box



R1: Deny if PATIENT_disagrees
R2: Permit if OR(HOSPITAL_agrees,SURGEON_agrees)

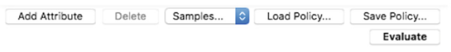
P: DOV(R1,R2)

(d) Policy



PATIENT_disagrees	True	False	[✓ Unknown]
HOSPITAL_agrees	True	[✓ False]	Unknown
SURGEON_agrees	[✓ True]	False	Unknown

(e) Attribute Values



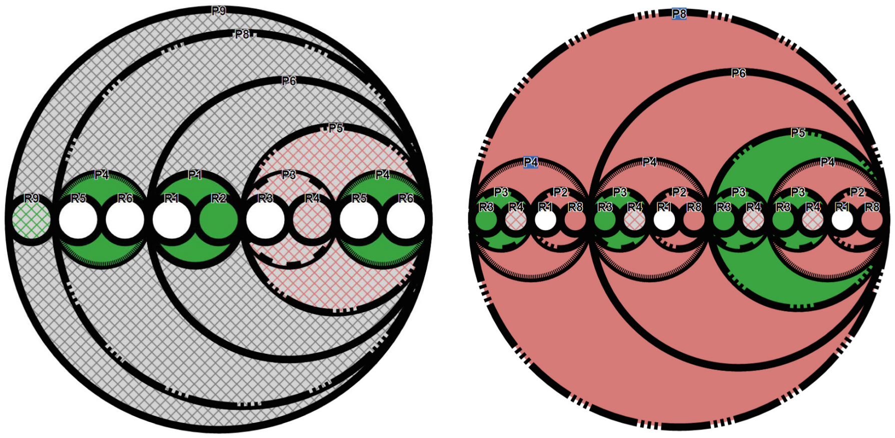
(f) VisABAC Controls

Fig. 5. (a) and (c) show the evaluation of the policy P, represented in (d), when attributes are set as (e) (fifth row of Table 1). The largest circle (P) is filled in with a grey pattern, since it evaluates to Indet(PD), the circle for R1 is filled in with a red pattern, since it evaluates to Indet(D), and the circle for R2 is filled in green, since it evaluates to Permit. (f) shows the screen section that provides common controls, such as Add Attribute, Delete Attribute, Samples, Load Policy, Save Policy and Evaluate. (Color figure online)

```
policyRules=
{"R1": "Permit if PATIENT_disagrees",
 "R2": "Permit if OR(HOSPITAL_agrees,
                     SURGEON_agrees)"
 "P": "DOV(R1,R2)"}
```

The *Attributes component* (Fig. 5(e)) allows the user to set the value for each attribute value: true, false, or unknown. For instance, Fig. 5(e) corresponds to a request where we do not know if the patient disagrees to the access, we know that the hospital does not agree to the access, and that the surgeon agrees, which corresponds to the fifth row of Table 1.

The *Decision component* (Fig. 5(c)) lists, for each rule in the Policy component, the decision obtained for that rule. These decisions are obtained by iterating through the `policyRules` object, following the evaluation rules established in [12]. The evaluation returns an object with the same structure, but where each rule has been replaced by its decision. In the case where a rule is not well-formed (e.g., missing reference, syntax error), it evaluates to Indeterminate(PD). Note that cycles in rule definitions are not currently detected, and an error would occur.



(a) Policy with 9 atomic rules and 9 binary policies. (b) Variation of 6(a) with sub-policies used multiple times.

Fig. 6. Circle packing visualisation of complex policies in VisABAC [26].

Finally, the *Visual component* (Fig. 5(a)) uses Zoomable Circle Packing to visually explore access control policies, using the D3.js library[10]. The zoomable aspect is a crucial one, as it allows the space occupied by the visualisation to remain constant. A *circle* is either a rule or a composition of rules grouped by a composition operators. As a consequence, a policy comprised of sub-policies is represented by circles containing sub-circles in a similar hierarchy as the given policy. The visual diagram is dynamic, and is updated when the policy or the attributes are updated and a new evaluation is calculated. Each circle is defined by two characteristics:

– The colour, which matches the result of the policy/rule they represent: green is for Permit, red for Deny, white for NA, patterned-green for Indet(P), patterned-red for Indet(D), and patterned-grey for Indet(PD). We have also developed a colour deficiency mode, which caters for different types of colour deficiencies. In addition, since these colours are set through a simple CSS (Cascading Style Sheet), they could be user configurable.

[10] https://d3js.org.

– The line pattern which matches the operator used. In particular, we use full lines for Deny-overrides and dashed lines for Permit-overrides. The lines for the other operators can be found in the online help of the tool.

For instance, Fig. 5(a) shows that *Level 0* (P) represents the whole policy by the most outer circle line; *Level 1* (R1 and R2) represent the first level of the tree policy with smaller circles inside. A zoom on the inner circles would display their respective targets, since they are atomic policies. Figure 6 illustrates more complex examples of ABAC policies.

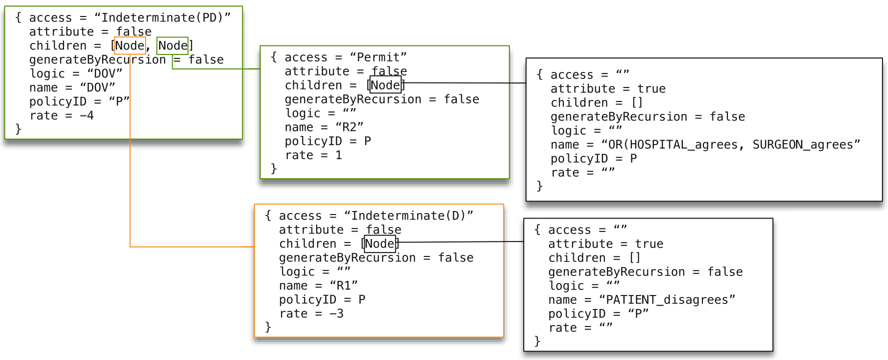
3.3 VisABAC Internals

VisABAC current version was coded in Javascript using NetBeans 8.2 on a multi-platform environment (macOS and Windows). Javascript was employed because it allows code transparency —source code could be easily explored and corrected by anyone who uses the application in a modern browser—. Consequently, the code is heavily commented and easily modifiable.

VisABAC follows standard web page creation conventions and, as such, it separates presentation elements description from the engine itself. All VisABAC code is located inside `visualiser/resources` and is categorised in `classes`, `images`, `libraries`, `pages`, `scripts` and `styles`.

Presentation. Almost all identifiable non-essential code is contained in the folders `images`, `pages`, `scripts` and `styles`. Files contained in each one of them are pretty much self explanatory and provide web page structure and non-essential elements (such as about, help, preferences, etc.); especial consideration is required only to the following items:

– `pages/VisualiserForm.html` contains the essential visual framework of the pilot VisABAC application and it could be modified to apply the engine to different products.
– `scripts/visualiserForm.js` contains the scripts that send messages to the VisABAC engine.
– `styles/logicCirclePacking.css` contains common styles used in the logic Circle Packing visualisation technique regardles of colour deficiency preferences; this is an essential VisABAC component.
– `styles/ visualiserForm.css`, `visualiserForm_ColorNormal.css` and `visualiserForm_ColorDeficiency.css` are used by `visualiser Form.html`.



```
{ access = “Indeterminate(PD)”
  attribute = false
  children = [Node, Node]
  generateByRecursion = false
  logic = “DOV”
  name = “DOV”
  policyID = “P”
  rate = -4
}
```

```
{ access = “Permit”
  attribute = false
  children = [Node]
  generateByRecursion = false
  logic = “”
  name = “R2”
  policyID = P
  rate = 1
}
```

```
{ access = “”
  attribute = true
  children = []
  generateByRecursion = false
  logic = “”
  name = “OR(HOSPITAL_agrees, SURGEON_agrees”
  policyID = P
  rate = “”
}
```

```
{ access = “Indeterminate(D)”
  attribute = false
  children = [Node]
  generateByRecursion = false
  logic = “”
  name = “R1”
  policyID = P
  rate = -3
}
```

```
{ access = “”
  attribute = true
  children = []
  generateByRecursion = false
  logic = “”
  name = “PATIENT_disagrees”
  policyID = “P”
  rate = “”
}
```

Fig. 7. Internal Tree structure formed by recurrent `Node` objects (`JSON` objects). The tree structure corresponds to the same sample shown in Fig. 5.

Engine. Essential code is contained in the folder `libraries` (only carrying `d3.v3.js`) and `classes`; the latest contains the following:

- Visualiser.js This is the class that creates the visualiser object. It stores all code that creates on screen the interface elements. It starts by `parsinPolicyRules` and storing as an internal attribute the policy passed by the user (`policyRules` and `policyAttributes`). The code also draws the screen components according to the preferences, picking the right colour mode.
- LogicCirclePacking_d3v3.js Very important class in which *zoomable circle packing* happens; it receives a `JSON` tree that represents the policy previously evaluated (Fig. 7 shows a sample representation). The evaluation are rates in the domain $[-5, -4, -3, -2, -1, 0, 1]$ to be corresponded by the D3 library into the range [indeterminateDColour, indeterminatePColour, indeterminatePDColour, indeterminateDColour, notApplicableColour, denyColour, permitColour]. These colours, as well as additional patterns are defined in this class to correctly represent permits, denies and indeterminations (indeterminate permit, indeterminate deny and indeterminate permit-deny), (Fig. 9 shows the indeterminate patterns). D3 uses a `svg` to plot the circle packing using very concise instructions, applying the same presentation function to all nodes of the tree almost simultaneously. Appearance functions are appended to lines as well and according to each `rate`, a particular stroke is use to draw a circle line. Figure 8 shows lines and its significance. This class also contains the zoom parameters required by the D3 library that allows policy navigation.
- Node.js is a class that specifies a non reducible element that recursively combined creates the policy tree. Figure 7 shows five samples of them forming a simple tree; most important attributes are:
  - `access` e.g. `Permit`, `Deny`, `IndeterminatePD`...
  - `attribute` used to mark if it is an attribute node `true` or `false`.
  - `children` array of dependant node operations.

  - `generateByRecursion` flag (`true` or `false`) to mark an auxiliar node created by a recursive evaluation.
  - `logic` stores composition operators, e.g. `DOV`, `POV`, `FA`, `OOA`, etc.
  - `name` e.g `DOV`, `R1`, etc. or any attribute name.
  - `policyID` unique identifier.
  - `rate` domain number resulting from node and its children evaluation.
- Policy.js This class provides methods and attributes to encapsulate all operations over a policy. It stores `policy`, `policyRules`, `policyAttributesByRule`, `policyRulesOrCompositions` and `policyTreeInJSON` and provides means to update them according to user interactions. The gist of the class are two main methods `resolveRules` and `parsePolicyToTree` which are called whenever there is an update or the program starts.
  `resolveRules` iterates through all policy rules (keys) to "solve" values, e.g.

```
policyRules={"PA": "Permit if attribute1",
             "PB": "Deny if attribute2",
             "PC": "DOV(PA,PB)"}
```

  will be transformed into:

```
_policy={"PA": "Permit",
         "PB": "Deny",
         "PC": "Permit"}
```

```
_policyAttributesByRule={"PA": "attribute1",
                         "PB": "attribute2",
                         "PC": "DOV(PA,PB)"}
```

  These two objects `_policy` and `_policyAttributesByRule` allow direct addressing either to results or attributes when `parsePolicyTree` is called. `resolveRules` also encapsulates a series of procedures (`resolveLogic`, `resolveRule`, `resolveRuleOrComposition`, `resolveAttributesByRule`) that handle the policy evaluation according to user inputting.
  `parsePolicyToTree` is the entry point to a series of methods, starting by `parseCompositionToTree` that creates a tree using the policy. `parseCompositionToTree` uses a series of stacks and recursion to evaluate fragments of the policy. Stacks have to be used in order to respect parenthesis hierarchy that might exist in a complex policy. Depending on complexity, also recursive procedures could be called. During this procedure, numeric rates are assigned to the nodes that are being created.
- Sample.js is a support class used to store attribute values and logic inside the Policy.js class.

4 Evaluation

VisABAC, presented in the previous section, is relatively easy to use, since it is defined as an in-browser application. The input language for policies is relatively straight-forward from an Attribute-based Access Control perspective.

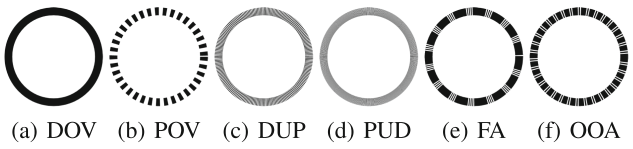


Fig. 8. Line conventions used by VisABAC to represent operations: (a) Deny overrides (b) Permit overrides (c) Deny unless permit (d) Permit unless deny (e) First applicable (f) Only one applicable. (c) and (d) may look similar in printing due to scale but they are clearly different in the application.

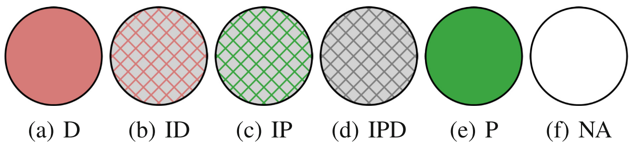


Fig. 9. Operations colour conventions used by VisABAC: (a) Deny (b) Indeterminate deny (c) Indeterminate permit (d) Indeterminate permit deny (e) Permit (f) Not applicable.

More importantly, the D3.js library for Circle Packing is particularly fluid, making the tool very responsive. Our participatory design elicited Circle Packing as the preferred visualisation technique, compared with other techniques such as foldable trees or treemaps. However, we are also interested in understanding whether VisABAC is effectively *usable*, i.e, whether its proposed graphical representation could help users in their tasks.

Nielsen and Levy argue that usability should be measured according to *subjective user preferences* and *objective performance measures*, since, in some cases, users have favoured interfaces that are measurably worse for them [32]. Similarly, MacLean et al. [23] found that subjects inclined towards a proven slower data entry method would still prefer it as long as it was not 20% slower than the faster method.

Hence, in addition to a subjective user preference questionnaire, we want to design an objective performance measure for using VisABAC. To the best of our knowledge, there is no standard benchmark for the usability of tools for access control policies, and therefore we define a new method in this paper. Roughly speaking, we give the user a fixed policy, a valuation for the attributes, and ask the user to change this valuation in order for the policy to evaluate to a specified decision. Our hypothesis is that the faster the user is able to do this task, the more they understand the policy, and thus the better is the tool with which the user interacts. We now describe this experimental settings, and we discuss the limitations of our approach in Sect. 6.

4.1 User Interface

We conduct a controlled-group experiment, where users in different groups see a different user interface. We define two different user interfaces (UI):

– The *Graphics UI* is an extension of the VisABAC interface, described in Sect. 3.2, with the addition of two main elements: the *context* box, which introduces the context of the policy, in English; and the *question* box, which specifies the expected decision. The boxes for the policy, the attributes, the decision box, and the visual decision diagram, are as described in Fig. 5[11].
– The *Text UI* is similar to the Graphics UI, as the notable exception that the visual decision diagram box is missing. However, the user still has access to the evaluation of the policy with the decision box.

4.2 Policy Question

The aim of either UI described above is to answer a question, given a context and a policy. Ideally, we would like to ask questions related to any aspect of the editing or maintenance of a policy. However, we believe that this would introduce too many different dimensions to control, and we focus instead on questions related to policy evaluation. We leave for future work the study of more complex questions. The context is a simple description of the motivation behind the policy, for instance, for the policy described in Sect. 2.1 and Fig. 5, the context is:

> *Releasing medical records in a certain hospital requires compliance with an access control policy. The system checks events with statements that return True or False if the forms have been filled and validated by the corresponding departments.*

The attribute values are initially set so that the policy evaluates to $\mathsf{Indet}(\mathsf{PD})$, and the question is:

> *Can you change the radio buttons so that PC evaluates to Deny?*

The user can change any radio button, and then click on a button *Evaluate*, which refreshes the different boxes with the new policy evaluation. There is no limit on the authorised number of evaluation per question, and they can go to the next question by clicking on the *Submit* button. They were also instructed they could go to the next question at any time if they did not wish to submit an answer for the current question, and this would be recorded as a wrong answer.

The experiment consists of a total of 32 sub-questions, grouped in 8 main questions. All sub-questions within a single main question have the same context, and only differ on minor details. For instance, a sub-question in the same group than the policy above use the First-Applicable (FA) operator to combine R1 and R2 instead of the Deny-Overrides (DOV). The main questions are denoted from Q1 to Q8, the sub-questions for the main question Q_i are denoted from Q_ia to Q_id.

[11] The full test with both interfaces is available from the front page of the tool.

4.3 Protocol

Each recruited participant P_i goes through the following steps:

1. After reading and signing the participant consent form, P_i is randomly assigned to either the *Text group* (the control group) or the *Graphics group* (the tested group).
2. P_i is presented with a short introduction about ABAC, going through a simple policy example (similar to that described in Sect. 2.1). At this stage, they can use the Text UI on the introduced example (the Graphics UI is only introduced in Step 4 for the Graphics group) and ask any question. They are also explained what is expected of them and informed that their time will be recorded. They are also informed that some policies are on purpose hard to analyse, and that we are measuring how the interface helps them, rather than assessing them. This step takes in average 10 min.
3. Once they feel confident about using the tool, they start answering the first series of main questions, Q1 and Q2 (8 sub-questions in total), using the Text UI, regardless of their assigned group.
4. After Q2, if P_i is in the Text group, they keep answering Q3 to Q8 (24 sub-questions in total); if P_i is in the Graphics group, they switch to the Graphics UI, and they are briefly introduced with the specifics of the Circle Packing representation; they then answer Q3 to Q8 using the Graphics UI.
5. After Q8, P_i is debriefed, and explained the purpose of the experiment. According to recommended practices [31], a £10 Amazon voucher is given as compensation for their time.

The entire protocol was designed to take, in average, between 30 to 45 min, including 20 min of actual assessment. The time to answer each question was visible to the participant, and although there was no strict countdown, to avoid adding time pressure, participants were encouraged to move on to the next question if they were spending more than 5 min on a sub-question (which happened in only one instance). The experiment took place in the same office and the same computer (a 27″ iMac), in order to control environmental changes. Participants were asked about colour deficiency, but none was indicated in our experiment.

4.4 Objective Performance Measure

Intuitively, we want to compare the time taken by users in the two different groups, in order to evaluate whether the Graphics UI was beneficial. However, performance measure among different individuals varies according to the capabilities of each one, and the nature of the experiment makes it hard to ensure the distribution of users in the groups is consistent with user capabilities. As a consequence, a procedure of normalisation had to be performed in order to compare data.

The selected normalisation value was the inverse of the number of seconds each participant spent on solving Q2 (i.e., the total time spent on subquestions

Q2a, Q2b, Q2c and Q2d). We denote this as the *normalisation coefficient* α_i, for each participant P_i. Subsequently, the time taken by P_i to answer each question is normalised by multiplying it by α_i. If this value is lower than 1, this implies the subject performed a particular question faster than Q_2 while a larger value represents the opposite. For instance, if P_1 took 4 s to complete Q2 ($\alpha_1 = 0.25$) and 6 s to complete Q3, their normalised time for Q3 is 1.5; if P_2 took 16 s to complete Q2 ($\alpha_2 = 0.0625$) and 23 s to complete Q3, their normalised time for Q3 is 1.4375. In other words, even though, absolutely speaking, P_2 was slower than P_1 for Q3, they were comparatively faster.

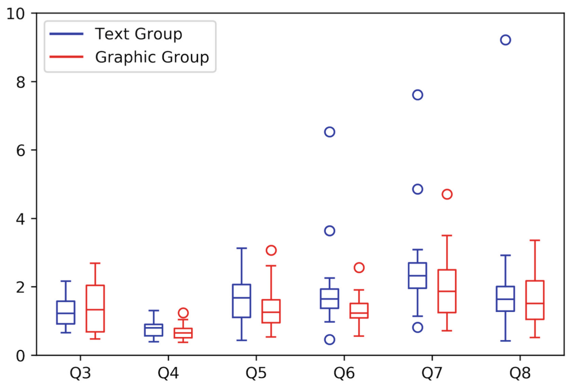


Fig. 10. Boxplots comparison of normalised times for questions Q3 to Q8 between the Text and Graphics groups (lower is better). The body of each box represents the intervals between the first (q_1) and third quartiles (q_3), the bar represents the mean, the whiskers represent the maximal and minimal values between $q_3 + 1.5(q_3 - q_1)$ and $q_1 - 1.5(q_3 - q_1)$, fliers represent points outside of this range.

This choice for the normalisation function comes from the fact that we have designed different questions with different levels of difficulty, Q7 being the most difficult for the full details of Q7). Hence, we expect that all users will spend more time to answer Q7 than Q2, and we want to measure this difference, rather than measuring directly the difference between users. Q2 was selected as the normalisation value since all participants, regardless of their group, had to do it with the Text UI, and it was assumed some familiarity was already gained by the user after performing Q1, since Q1 and Q2 have a similar complexity level.

4.5 Subjective User Preferences

Subjective Testing was performed on users who were exposed to the visualisation technique. A relatively standard questionnaire was presented to collect their impressions using a *Likert scale* [32] after finishing the objective testing.

5 Results

We recruited 32 participants over 4 weeks, mostly among undergraduate Computer Science students, with no formal knowledge of ABAC, and randomly assigned to the groups (16 participants each). The aim of this study was to assess the impact of circle packing, so we targeted a relatively uniform group in terms of prior knowledge, rather than experts in Access Control. Figure 10 shows the normalised time average of participants for each question, including wrong answers (there are 8 wrong answers in each group). The mean for the Graphics group is lower (i.e., better) from Q4 to Q8 (comparatively to the time taken for Q2) compared to the Text group. The mean of Graphics group is higher for Q3, which could indicate a small learning curve with the Graphics UI.

Altogether, the normalised mean time for participants in the Text group to answer all questions from Q3 to Q8 is $m_t = 10.38$ (with a confidence interval of $[7.88, 12.88]$ and a standard deviation of $\sigma_t = 5.10$). In comparison, the normalised mean time for participants in the Graphics group is $m_g = 8.58$ (with a confidence interval of $[7.33, 9.83]$ and a standard deviation of $\sigma_g = 2.55$). This allows us to conclude that the effect size[12] is 0.44, which is traditionally seen as a small to medium effect size [10].

In addition, the results of the user preferences survey showed that 82.35% of participants described the presence of the visualisation as useful; 76.47% of participants felt more confident operating the policy with the presence of the graph and 47.06% agree and 35.39% agree to some extent that the presence of the graph makes them feel they understand the policy better. Some questions were however very conclusive, e.g. if complex mental operations were needed, which could indicate this question was not well formulated.

6 Conclusions

Building a usable visualisation tool for access control policies is a challenging task, as it requires: (i) to have a good understanding of the existing literature on visualisation; (ii) to be based on a clear semantics for the access control language; (iii) to use a participatory design process; (iv) to be validated with a user study. We have successfully followed these steps in the design of VisABAC, which is the first visualisation tool for attribute-based access control policies, where composition operations seems to be adequately represented and details are disclosed on demand thanks to the zooming and progressive disclosure of tags. VisABAC also provides interactivity to the user and increments the exploring of the policy in a graphical manner. The extensive literature survey presented in Sect. 2.2 is, to the best of our knowledge, the first survey on visualisation technique for access control policies.

The participatory design process was positive, and most users liked the concept very much, found it intuitive and easy to use, although they remarked that

[12] Cohen's effect is computed as $(m_t - m_g)$ divided by $\sqrt{(\sigma_t^2 + \sigma_g^2)/2}$.

some training could have decrease their response time. Crucially, the experiment showed a small to medium effect size [26], allowing us to conclude that VisABAC improves the handling of attribute-based access control policies for a population with no formal training. Of course, at this stage, it is not yet clear whether VisABAC can provide a significant contribution to access control experts, but we believe the tool as presented here and our results pave the way towards an experiment at a larger scale.

Future Work. VisABAC is specifically designed to be open and easy to extend. The underlying infrastructure uses HTML (for the basic interface), JSON (for the encoding of the policies), and Javascript (for the evaluation of policies and the visualisation elements), making it possible to consider other visualisation techniques. In particular, the collapsible tree approach (see Sect. 3.1) has received some positive response during the participatory design phase of VisABAC (policies tend to be naturally seen as trees), but suffers from a space occupation issue. The textual input for VisABAC can also be straightforwardly extended, for instance by parsing directly XACML policies, making it possible to compare real XACML cases against their visualisation (and not synthetic ones), and include authoring tools such as VisPE [29].

Acknowledgements. This work was partially supported by the UK National Centre for Cyber-Security, in the context of the Research Institute in the Science of Cyber-Security. The authors would also like to thank Nick Holliman from Newcastle University for very useful discussions on visualisation techniques.

References

1. Alavi, R., Islam, S., Mouratidis, H.: A conceptual framework to analyze human factors of Information Security Management System (ISMS) in organizations. In: Tryfonas, T., Askoxylakis, I. (eds.) HAS 2014. LNCS, vol. 8533, pp. 297–305. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07620-1_26
2. Ashley, P., Hada, S., Karjoth, G., Powers, C., Schunter, M.: Enterprise privacy authorization language (EPAL). IBM Research (2003)
3. Barker, S.: The next 700 access control models or a unifying meta-model? In: SACMAT, pp. 187–196. ACM (2009)
4. Barrett, R., Kandogan, E., Maglio, P.P., Haber, E.M., Takayama, L.A., Prabaker, M.: Field studies of computer system administrators: analysis of system management tools and practices. In: Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work, CSCW 2004, pp. 388–395 (2004). https://doi.org/10.1145/1031607.1031672
5. Bastian, M., Heymann, S., Jacomy, M.: Gephi: an open source software for exploring and manipulating networks. In: Third International AAAI Conference on Weblogs and Social Media (2009)
6. Bauer, L., Garriss, S., Reiter, M.K.: Detecting and resolving policy misconfigurations in access-control systems. In: SACMAT, pp. 185–194. ACM (2008)

7. Becker, J., Heddier, M., Öksüz, A., Knackstedt, R.: The effect of providing visualizations in privacy policies on trust in data privacy and security. In: 2014 47th Hawaii International Conference on System Sciences, pp. 3224–3233 (2014). https://doi.org/10.1109/HICSS.2014.399
8. Benantar, M.: Access Control Systems: Security, Identity Management and Trust Models. Springer, Boston (2005). https://doi.org/10.1007/0-387-27716-1
9. Card, S.K., Mackinlay, J.D., Shneiderman, B. (eds.): Readings in Information Visualization: Using Vision to Think. Morgan Kaufmann Publishers Inc., San Francisco (1999)
10. Cohen, J.: Statistical Power Analysis for the Behavioral Sciences, pp. 20–26. Lawrence Earlbaum Associates, Hillsdale (1988)
11. Crampton, J., Morisset, C.: PTaCL: a language for attribute-based access control in open systems. In: Degano, P., Guttman, J.D. (eds.) POST 2012. LNCS, vol. 7215, pp. 390–409. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28641-4_21
12. Crampton, J., Morisset, C., Zannone, N.: On missing attributes in access control: Non-deterministic and probabilistic attribute retrieval. In: SACMAT, pp. 99–109. ACM (2015)
13. Euler, L.: Lettres a une princesse d'allemagne. Sur divers sujets de physique et de philosophie, vol. 2. Birkhauser, Basel (1761)
14. Fisler, K., Krishnamurthi, S., Meyerovich, L.A., Tschantz, M.C.: Verification and change-impact analysis of access-control policies. In: Proceedings of the 27th International Conference on Software Engineering, ICSE 2005, pp. 196–205. ACM, New York (2005). https://doi.org/10.1145/1062455.1062502
15. Heydon, A., Maimone, M.W., Tygar, J.D., Wing, J.M., Zaremski, A.M.: Miro: visual specification of security. IEEE Trans. Softw. Eng. **16**(10), 1185–1197 (1990). https://doi.org/10.1109/32.60298
16. Johnson, B., Shneiderman, B.: Tree-maps: a space-filling approach to the visualization of hierarchical information structures. In: Proceedings of the 2nd Conference on Visualization 1991, Los Alamitos, CA, USA, pp. 284–291. IEEE (1991)
17. Kirlappos, I., Sasse, M.A.: What usable security really means: trusting and engaging users. In: Tryfonas, T., Askoxylakis, I. (eds.) HAS 2014. LNCS, vol. 8533, pp. 69–78. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07620-1_7
18. Kolovski, V.: Logic-based access control policy specification and management. Technical report, Department of Computer Science, University of Maryland, College Park (2007)
19. Kordon, F.: An introduction to rapid system prototyping. IEEE Trans. Softw. Eng. 28(9), 817–821 (2002). https://doi.org/10.1109/TSE.2002.1033222
20. Lacey, D.: Managing the Human Factor in Information Security: How to Win over Staff and Influence Business Managers. Wiley, Hoboken (2009)
21. Lampson, B.W.: Protection. Oper. Syst. Rev. **8**(1), 18–24 (1974). https://doi.org/10.1145/775265.775268
22. Licht, D.M., Polzella, D.J., Boff, K.R.: Human factors, ergonomics and human factors engineering: an analysis of definitions. Crew System Ergonomics Information Analysis Center (1989)
23. MacLean, A., Barnard, P., Wilson, M.: Evaluating the human interface of a data entry system: user choice and performance measures yield different tradeoff functions. People Comput. Des. Interface **5**, 45–61 (1985)
24. Meyer, M.: Information visualization for scientific discovery, April 2011. https://www.youtube.com/watch?v=Sua0xDCf8MA

25. Montemayor, J., Freeman, A., Gersh, J., Llanso, T., Patrone, D.: Information visualization for rule-based resource access control. In: Proceedings of International Symposium on Usable Privacy and Security (SOUPS), p. 24 (2006)
26. Morisset, C., Sanchez, D.: VisABAC: a tool for visualising ABAC policies. In: Proceedings of the 4th International Conference on Information Systems Security and Privacy - Volume 1: ICISSP, pp. 117–126. INSTICC, SciTePress (2018). https://doi.org/10.5220/0006647401170126
27. Morisset, C., Zannone, N.: Reduction of access control decisions. In: SACMAT, pp. 53–62. ACM (2014)
28. Mousas, A.S., Antonakopoulou, A., Gogoulos, F., Lioudakis, G.V., Kaklamani, D.I., Venieris, I.S.: Visualising access control: the prism approach. In: 2010 14th Panhellenic Conference on Informatics (PCI), pp. 107–111, September 2010. https://doi.org/10.1109/PCI.2010.52
29. Nergaard, H., Ulltveit-Moe, N., Gjøsæter, T.: ViSPE: a graphical policy editor for XACML. In: Camp, O., Weippl, E., Bidan, C., Aïmeur, E. (eds.) ICISSP 2015. CCIS, vol. 576, pp. 107–121. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-27668-7_7
30. Ngo, C., Makkes, M.X., Demchenko, Y., de Laat, C.: Multi-data-types interval decision diagrams for XACML evaluation engine. In: 2013 Eleventh Annual International Conference on Privacy, Security and Trust (PST), pp. 257–266, July 2013. https://doi.org/10.1109/PST.2013.6596061
31. Nielsen, J.: Usability Engineering. Morgan Kaufmann Publishers Inc., San Francisco (1993)
32. Nielsen, J., Levy, J.: Measuring usability: preference vs. performance. Commun. ACM **37**(4), 66–75 (1994). https://doi.org/10.1145/175276.175282
33. Pan, L., Liu, N., Zi, X.: Visualization framework for inter-domain access control policy integration. China Commun. **10**(3), 67–75 (2013). https://doi.org/10.1109/CC.2013.6488831
34. Pan, L., Xu, Q.: Visualization analysis of multi-domain access control policy integration based on tree-maps and semantic substrates. Intell. Inf. Manag. **4**(5), 188–193 (2012)
35. Pina Ros, S., Lischka, M., Gómez Mármol, F.: Graph-based XACML evaluation. In: Proceedings of the 17th ACM Symposium on Access Control Models and Technologies, SACMAT 2012, pp. 83–92. ACM, New York (2012). https://doi.org/10.1145/2295136.2295153
36. PwC: 2015 information security breaches survey. Technical report, HM Government and PwC Consulting and Infosecurity Europe, April 2015
37. Rao, P., Ghinita, G., Bertino, E., Lobo, J.: Visualization for access control policy analysis results using multi-level grids. In: IEEE International Symposium on Policies for Distributed Systems and Networks, pp. 25–28 (2009). https://doi.org/10.1109/POLICY.2009.29
38. Riche, N.H., Dwyer, T.: Untangling Euler diagrams. IEEE Trans. Vis. Comput. Graph. **16**(6), 1090–1099 (2010). https://doi.org/10.1109/TVCG.2010.210
39. Rissanen, E., Lockhart, H., Moses, T.: XACML V3.0 administration and delegation profile version 1.0. Committee Draft 1 (2009)
40. Ritter, F.E., Baxter, G.D., Churchill, E.F.: Foundations for Designing User-Centered Systems. Springer, London (2014). https://doi.org/10.1007/978-1-4471-5134-0
41. Rodgers, P.: A survey of Euler diagrams. J. Vis. Lang. Comput. **25**(3), 134–155 (2014). https://doi.org/10.1016/j.jvlc.2013.08.006

42. Rosa, W.D.: Toward visualizing potential policy conflicts in eXtensible Access Control Markup Language (XACML). Theses and dissertations, University of New Orleans, New Orleans, May 2009
43. Sackmann, S., Kähmer, M.: ExPDT: Ein policy-basierter ansatz zur automatisierung von compliance. Wirtschaftsinformatik **50**(5), 366–374 (2008)
44. Sato, Y., Mineshima, K., Takemura, R.: The efficacy of Euler and Venn diagrams in deductive reasoning: empirical findings. In: Goel, A.K., Jamnik, M., Narayanan, N.H. (eds.) Diagrams 2010. LNCS (LNAI), vol. 6170, pp. 6–22. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14600-8_6
45. OASIS Standard: eXtensible Access Control Markup Language (XACML) version 2.0 (2005)
46. Stapleton, G., Zhang, L., Howse, J., Rodgers, P.: Drawing Euler diagrams with circles. In: Goel, A.K., Jamnik, M., Narayanan, N.H. (eds.) Diagrams 2010. LNCS (LNAI), vol. 6170, pp. 23–38. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14600-8_7
47. Stepien, B., Matwin, S., Felty, A.: Strategies for reducing risks of inconsistencies in access control policies. In: 2010 International Conference on Availability, Reliability and Security, pp. 140-147 (2010)
48. Trudeau, S., Sinclair, S., Smith, S.W.: The effects of introspection on creating privacy policy. In: WPES 2009: Proceedings of the 8th ACM Workshop on Privacy in the Electronic Society, pp. 1–10. ACM, New York (2009). https://doi.org/10.1145/1655188.1655190
49. Vaniea, K., Ni, Q., Cranor, L., Bertino, E.: Access control policy analysis and visualization tools for security professionals. In: SOUPS Workshop (USM) (2008)
50. Wang, W., Wang, H., Dai, G., Wang, H.: Visualization of large hierarchical data by circle packing. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI 2006, pp. 517–520. ACM, New York (2006). https://doi.org/10.1145/1124772.1124851
51. Xu, W., Shehab, M., Ahn, G.J.: Visualization based policy analysis: case study in SELinux. In: Proceedings of the 13th ACM Symposium on Access Control Models and Technologies, SACMAT 2008, pp. 165–174. ACM, New York (2008). https://doi.org/10.1145/1377836.1377863