JustAct: Actions Universally Justified by Partial Dynamic Policies

Christopher A. Esterhuyse ✉[1,2], Tim Müller[1,2], and
L. Thomas van Binsbergen[1,3]

[1] Informatics Institute, University of Amsterdam, The Netherlands
[2] { ✉ c.a.esterhuyse,t.muller}@uva.nl
[3] ltvanbinsbergen@acm.org

Abstract. Inter-organisational data exchange is regulated by norms originating from sources ranging from individual consent to (inter)national laws. Verifying norm-compliance is complex because laws (e.g., GDPR) distribute responsibility and require accountability. Moreover, in some domains (e.g., healthcare), the norms themselves may be private. In contrast, standard solutions (e.g., access- and usage-control, smart contracts) reason about policies that are assumed to be public. Instead, we present a novel framework prescribing how decentralised agents decide which actions are justified, despite their partial views of the policy. Crucially, justifications are universal, e.g., accepted by future auditors. Agents establish a common notion of compliance through an (externally synchronized) agreement, which is the basis of each justification defined by policy fragments agents autonomously create, gossip, and assemble. We demonstrate our framework with a federated medical data processing system, using Datalog with weak negation as a minimal policy language.

Keywords: Decentralised · Framework · Composition · Coordination · Multi-agent System · Policy · Program Refinement · Specification .

1 Introduction

Data exchange systems are distributed systems facilitating the controlled sharing, trading, and processing of (often large) datasets and analysis results within data exchange applications, increasing the public, commercial or academic value of collected data. Following the inter-organisational nature of data exchange systems, and the (market or privacy) sensitive nature of the exchanged assets, collaborating organisations adopt (potentially complex) governance models [53] in an attempt to ensure compliance with regulations and contractual agreements. In support of such governance models, high levels of control should be given to organisations to influence the execution of data exchange applications, e.g. via access control [40,44] or usage control [25,36,59]. Furthermore, high levels of accountability are required

to support dispute resolution [48] and for demonstrating legal compliance [14]. Data exchange systems exhibit a fundamental trade-off between maximising the availability of data to data users and maximising the control over data to data owners, subjects and (privacy) authorities. In this work we present a framework that enables the organisations collaborating in a data exchange system to formalise their shared and individual position with respect to this trade-off through powerful, declarative *policies*.

Our approach is to define a framework that specifies the relation between runtime system dynamics (messages and actions) and statics (policies and facts) such that they interact. In one direction: agent messages create and disseminate policies. In the other direction: policies specify which actions are permitted. Crucially, permission is decidable, despite agents having only partial knowledge of the existing policies, and despite policies being changed at runtime. Moreover, these decisions are agreed by all agents, e.g., an actor can be confident that other actors (e.g., auditors) will agree that their actions were permitted. For maximum applicability, our framework is parametric to the *policy language*, whose syntax defines the set of *policies*, and whose semantics defines their relation to *facts*. In this work, we demonstrate the framework as instantiated with the policy language *Datalog with weak negation* (Datalog¬).

This work represents a step in an on-going investigation into generic, policy-driven data exchange systems satisfying legal requirements (e.g. accountability and auditability). The framework in this paper has a prototype implementation supported by a bespoke domain-specific policy language (not presented here). We intend to make our policy framework an integral part of the Brane workflow execution system [55], a central component in the EPI Framework [26], and demonstrate its applicability in a variety of use cases.

After some background (Section 2), we contribute:

1. the definition of a **framework** for agents acting on shared policies while communicating, refining, and modifying those policies (Section 3),
2. a **demonstrative application** of our instantiated framework to a distributed, multi-agent, medical workflow processing system (Section 4).

In Section 5, we consider implementation decisions for instantiating the framework. We discuss our contributions by their own merits (Section 6) and in comparison to related work (Section 7) before concluding with a summary (Section 8).

2   Background

Distributed Systems and Algorithms *Distributed systems* model the distribution of a stateful *configuration* over a set of *processes*; each process has its own, local *state*. *Distributed algorithms*, when implemented by each processes, give systems useful, emergent properties. Often, these algorithms assume only basic, asynchronous and unreliable message-passing, comparable to IP and UDP. Some algorithms solve distributed problems (e.g., self-stabilisation), and create useful abstractions (e.g., synchronisers).
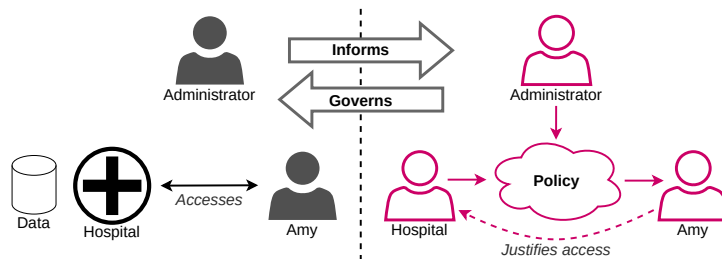
Fig. 1. Conceptual use-case of the framework. Agents autonomously choose to act, accessing data in the real world (left), as permitted by their shared policies (right).

In this work, we refer to two classes of algorithms. Firstly, *gossip* protocols disseminate information from peer (process) to peer, which results in decentralisation and robustness, by imposing minimal requirements on the network topology and process behaviour [5]. Secondly, *consensus algorithms* establish fundamental agreement on the selection of a value, consistently among processes. Consensus has been well-studied for decades [41], but has seen renewed interest in application to blockchain technologies, for example, in [29, 30].

Agents and Autonomy The field of *multi-agent systems* studies processes (called *agents*) that exhibit social phenomena as a result of their agency or *autonomy*: agents are motivated by goals to draw from their partial information to act on shared resources and interact with other agents. Literature explores a variety of (software models of) social organisations, ranging from cooperative data-sharing *consortia* (e.g., in [19]) to competitive markets (e.g., in [61]).

Agent-oriented programming studies the programming of agents, balancing the usual software- and language-engineering concerns, shared with object-oriented programming, with a unique emphasis on agent autonomy. For example, agent autonomy also tends to improve system scalability and robustness. All these ideas are present in seminal agent-oriented programming works like [51], and persist into more recent works like the survey [34].

Policy Languages We give a very brief overview of the various forms of policy developed in various disciplines, and influencing our work.

Access control is a mainstay in cyber-physical systems that revolves around the regulation of events of agents accessing resources. Policies often take the form of conditional rules [20, 45], sometimes applied in the context of meta-data attributes [47]. *Usage control* generalises access control such that access events occur for a duration of time; access must be maintained, and can be interrupted [1]. XACML [2] and ODRL [24], are particular policy languages for implementing access- and usage-control. For example, [54] implements usage control in XACML. The languages differ in the details. For example, XACML maps stages of the enforcement pipeline to agent roles: policy-administration-, -decision-, -enforcement-, -information-, and -retrieval-points.

Normative specifications specify fundamental social relations such as power, duties, rights, obligations, and permissions [3]. Legal regulations specify normative policies ("norms") in the context of executive (e.g., governmental) agencies and organisations. The EU General Data Protection Regulation [18] specifies the legal usage and access to data within the European Union. Its wide reach and impact make it influential even outside the EU. The study of norms reflects its long history in its rich nomenclature, for example, [10] clarifies the relationship between *substantive* and *procedural* norms, and [23] defines *open-texture terms*.

A wealth of other works intentionally blurs the line between these various notions of policy. For example, the eFLINT language [7] formalises norms using the Hohfeldian framework of legal proceedings [58] and has been used for access control [6]. Symboleo [49] and Fievel [57] are similar languages with similar goals, that differ in the details. For example, eFLINT particularly emphasises its logical reasoning features. These tools afford the application of various disciplines and tools to policies. For example, applying model-driven development [46], and model-checking for high-level properties in policies, e.g., expressed in Symboleo [38].

The Datalog¬ Language *Logic programming* languages are designed to straight-forwardly operationalise various logics. Here, we give an account of Datalog¬ sufficient to understand the Datalog¬ examples throughout this article.

Datalog, overviewed in [12], is a simple logic programming language: each program is a set of *Horn clauses* called *rules*. Precisely, each rule has form ($c_1 \wedge c_2 \wedge ... \wedge c_m \leftarrow a_2 \wedge a_2 \wedge ... \wedge a_n$), where *consequents* $c_{1...m}$ and *antecedents* $a_{1...n}$ are *facts* constructed by applying a predicate symbol p to constants and (first-order) variables. The Datalog semantics gives each program a *model*, mapping *ground* facts (without variables) to Boolean values. Most literature uses the same concrete syntax: ($\leftarrow$) and ($\wedge$) are denoted (`:-`) and (`,`), respectively, and only variable identifiers begin with uppercase letters. For example, `knows(amy,X) :- knows(X,amy)` formalises "Amy knows everyone that knows Amy".

Various dialects of Datalog have been studied in the literature, exploring the combination of various features. Datalog¬ [50] is a useful generalisation: antecedents may be negated (with ¬, often written `not`), conditioning consequents on the absence of truths. This strictly improves expressiveness [28], because it affords *non-monotonic reasoning*: each reasoning step may *remove* truths [52]. Equivalently, truth is non-monotonic with respect to the addition of rules to programs. For example, fact `sun` is true in program `sun :- not clouds`, but false after rule `clouds` is added; we say `sun` is *falsified*. Unfortunately, not all Datalog¬ programs have unique logical interpretations. Accordingly, different semantics exist (e.g., stable model [22] and well-founded [56]) attributing different models to these *unstratified* (defined in [42]) programs. For example, what should be the value of p in `p :- not p`? Fortunately, we consider no such programs in this article.

Several tools can interpret (super-languages of) Datalog¬. For example, the Clingo answer-set solver [21][4] can interpret each Datalog¬ example in this article.

---

[4] An online Clingo interpreter is available at `https://potassco.org/clingo/run`

3 Distributed Runtime Framework

This section defines the framework by supplementing Figure 2 with requirements, explanations, motivations, and examples. We summarise the framework as follows:

Agents make statements carrying policies that model the system.
Agents gossip, assemble, and validate statements to build justifications.
Agents only take actions that are permitted by their justifications.
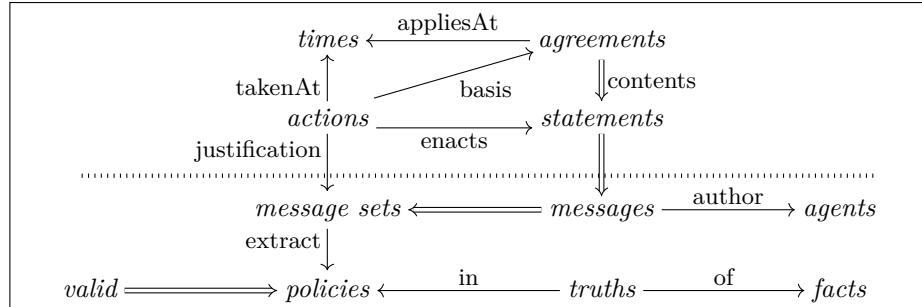

Fig. 2. Graph of sets and functions (objects) defined by users of the framework. Sets are named in italics. Functions are denoted as arrows ($\rightarrow$) from domain to co-domain. Functions are identified by their co-domain (or by a label if given). The dotted line partitions objects into *dynamics* (above) and *statics* (below). At runtime, new elements may be added to dynamic objects, but static objects are fixed. Each ($\Rightarrow$) is a typed identity function, e.g., *statements* $\subseteq$ *messages*. All functions shown are pure and total. For example, some policy is extracted from each message set.

Figure 2 defines a set of named *objects*: sets and functions over the sets. The framework is used by defining (the elements of) these objects. We partition objects into *static* and *dynamic* objects, each discussed in their own subsections. To use the framework, a user *instantiates* it by defining the objects in Figure 2 such that the *framework requirements* are preserved. The most fundamental framework requirements are expressed in the figure; for example, each *action* has exactly one *justification*, and that each *agreement* is a *message*. The remaining framework requirements are introduced throughout this section. The framework guarantees Properties 1 and 2, i.e., agents always agree on action *permission* and *effects*, which are both precisely defined in Section 3.2.

Ultimately, we explain how agents can act autonomously while enforcing the *well-behavedness* (Definition 1) of themselves and their peers in practice.

Property 1 *Agents always agree whether a given action is permitted.*

Property 2 *Agents always agree on the effects of a given action.*

Definition 1 *An agent is well-behaved iff all of its actions is permitted.*

The rest of the section discusses the framework in general, and incrementally builds an example framework instantiation, using policies built from only the the Datalog$^\neg$ rules in Table 1. We use a running example of well-behaved agents Amy and Bob. Ultimately, Bob enacts r_d, which has the effect of deleting Data1.

Henceforth, we use the following notation. We distinguish definitions ($\triangleq$) from assertions of equality ($=$). We denote disjoint set union as ($\uplus$), i.e., $a \uplus b = a \cup b$ iff sets a and b are disjoint, and otherwise, $a \uplus b$ is undefined.

3.1   Static Objects

The static objects are fixed at runtime. They concern the fundamentals of the framework: data types, and the syntax and semantics of the policy language.

The **agents**[5] of the framework author **messages** occurring dynamically as statements. We take for granted that ($\in$) relates messages to *message sets* as expected, and we treat each message m also as the singleton set $\{m\}$. Statements communicate **policies** (see **extract** below). Policies determine which subset of possible facts are *true*, defined $true(f, p) \triangleq \exists t \in truths : (of(t) = f) \wedge (in(t) = p)$. We instantiate *policies* as the subsets of the *(policy) rules* shown in Table 1. Hence, ($\cup$) is a natural policy-composition operator, unifying rules. Note that composition does not generally unify *truths*. We use the semantics of Datalog$^\neg$ to define facts and truth: f is true in p iff $f \in [p]$, the *stable model* of policy p, for example, it holds in the case where $f = \texttt{error}$ and $p = \{r_m\}$.

We instantiate **valid** to characterise policies without errors. Precisely, we let $p \in valid \triangleq \neg true(\texttt{error}, p)$. Despite the simplicity of this definition, (in)validity emerges from complex rule interactions. For example, no valid policy is a superset of $\{r_m, r_n\}$, as these suffice to infer the truth of $\texttt{error}$. Section 4 demonstrates how (in)validity lays the groundwork for inter-agent power dynamics.

The definition of **extract** determines the policies agents can express, as agents understand policy $extract(m)$ as the subjective assertions of agent $author(m)$.

We instantiate *extract* as a pure function of the message author and the message *payload*, which is an arbitrary policy, chosen by the author. Intuitively, *extract* invalidates payloads with undesirable author-rule pairs. Precisely, we let $extract(\{m\}) \triangleq payload(m) \cup \{\texttt{error} \mid \forall (r, a) \in owns : (r \in m \wedge \neg author(m, a))\}$,

----

[5] Unless otherwise specified, we let framework agents coincide with the agents of the underlying distributed system.

Name	In natural language	As Datalog$^\neg$ rules
r_m	Amy <u>must</u> confirm.	`error :- not ctl-confirms(amy)`
r_n	Amy must <u>not</u> confirm.	`error :- ctl-confirms(amy)`
r_c	Amy <u>confirms</u> if someone is trusted.	`ctl-confirms(amy) :- ctl-trusts(amy, X)`
r_t	Amy <u>trusts</u> Bob.	`ctl-trusts(amy, bob)`
r_d	Bob <u>deletes</u> Data1.	`ctl-deletes(bob, data1)`

Table 1. Example *(policy) rules* expressed in Datalog$^\neg$ and natural language. Each rule's name is suggested by the underlined keyword. A policy is any rule set.

where *owns* relates rule r to agent a if r has an antecedent whose predicate is prefixed by `ctl-` and whose first parameter is constant a. For example, only Bob owns rule `ctl-deletes(bob, data1)`, while rule `error :- confirms(amy)` has no owner because fact `error` has neither prefix `ctrl-` nor a constant first parameter. Finally, we let $extract(\emptyset) \triangleq \emptyset$, $extract(\{m\} \uplus M) \triangleq extract(\{m\}) \cup extract(M)$, i.e., the policy extracted from each message set is the composition of policies extracted from each message. This simple definition confers a property that simplifies reasoning: *extract* commutes and associates over message set union.

The above definition of *extract* uses invalidity to constrain which subjective assertions are available to each agent. For example, because Bob owns r_d, for each message m, either Bob authors m, $r_d \notin payload(m)$, or $extract(m)$ is invalid. Bob can rest assured that Amy alone cannot meaningfully assert r_d.

3.2   Dynamic Objects

Dynamic objects are defined by each system configuration at runtime. They instantiate the statics, and must be stored and communicated by agents.

At runtime, a subset of messages are **statements**, the messages that have been created and shared at runtime. Intuitively, statements have few restrictions, but therefore, they have little impact of their own. They are meaningful because of their relationship with actions. In practice, we expect each statement to be *autonomously* created by its author, i.e., as decided by the author alone. In theory, the set of statements grows forever. In practice, agents and networks have finite storage capacity, so we let agents forget or discard statements. For our purposes, it suffices if the existence of a given statement is semi-decidable: agents can decide (and prove) that it exists by observing (and showing) it. In practice, the integrity and provenance of statements is preserved by authors cryptographically signing their statements, and agents ignoring statements without their authors' signatures. Agents share statements with their peers (e.g., via gossip), for their own reasons, and at their own pace. Section 4 demonstrates a case where an agent intentionally withholds a statement from other agents.

Agreements attribute special meaning to selected statements at selected times. We require that membership be decidable. Intuitively, this lays the groundwork for agents agreeing which actions are permitted. In practice, agents explicitly synchronise the set of agreements. Synchronisation may be infrequent; e.g., one synchronisation creates agreements for times from 100 to 500. Synchronisation may be sparse; e.g., agents maintain a synchronised *current* time and implicitly extend all current agreements until the next explicit synchronisation. For simplicity in the examples henceforth, we assume a set of *current* agreements which appears to change as different statements are agreed to apply at future times.

At runtime, there is a growing set of (taken) **actions**. Each action is *takenAt* some time, has a *basis* agreement, *enacts* a statement, and is *justified* by a message set. We call agent a the *actor* of action a' iff $a = author(enacts(a'))$. Like statements, actions are created autonomously (by their actors) and then shared between agents by unspecified gossip. Unlike statements, actions are restricted such that they can be effectful and meaningful. Precisely, well-behaved

agents only *permitted* actions, defined as preserving Properties 3 to 6. Well-behavedness has only extrinsic value: when an agent takes a non-permitted action, this violates the expectations and trust of their peers, and may have external consequences. For example, Amy is harmed if Bob deletes Data1 without permission. Well-behavedness is enforceable if Properties 3 to 6 are each decidable.

In practice, agents preserve their own well-behavedness with *ex-ante* enforcement, i.e., checking permission *before* acting. Agents enforce well-behavedness on each other using *ex-post* enforcement, i.e., checking permission of taken actions via run-time monitoring or auditing. For example, we expect real systems to require agents to remain accountable of their actions to auditors, such that they cannot preserve the appearance of well-behavedness by hiding their actions.

Property 3 (Stated) $\forall a \in actions : justification(a) \subseteq statements.$

Property 4 (Relevant) $\forall a \in actions : \{basis(a), enacts(a)\} \subseteq justification(a).$

Property 5 (Valid) $\forall a \in actions : extract(justification(a)) \in valid.$

Property 6 (Based) $\forall a \in action : takenAt(a) = appliesAt(basis(a)).$

Property 3 requires that justifications consist only of statements. For example, where $author(m) = $ Bob, until Bob states m, Amy cannot take actions which necessarily include m in their justification. Property 4 ensures that each justification includes the action's basis and enacted statements. Thus, each justification is constrained by some agreement, and by the enacted statement. For example, Bob cannot enact r_d with a justification in which `deletes(bob, data1)` is not true. Property 5 limits justifications to those whose extracted policies are valid. For example, recall that only Bob owns r_d. Hence, Amy can make statements whose payloads include r_d, but those statements can be safely ignored, as they can never be included in valid justifications. Finally, Property 6 ensures that each action is based on an agreement that applies at the time the action is taken. Clearly, adding new agreements (e.g., to future times) does not affect past actions. However, the choice of agreements that apply in the future determines which actions are permitted in the future. For example, the selection of future agreements simulates updating the payload of the currently agreed statement m_a from $\{r_m, r_t\}$ to $\{r_t\}$, mirroring relaxations in the requirements of the GDPR. New actions are permitted to Bob in the future, as Amy's confirmation is no longer required.

Property 1 follows from agents coming to the same conclusions about the permission of each action a, as Properties 3 to 6 depend only on a's justification and the agreement when it is created (which both remain fixed). Property 2 follows from fixing the effects of each action. We propose the following definition in particular: $effects(a) \triangleq \{f \mid \forall f \in facts, true(extract(enacts(a)), f)\}$, i.e., each action's effects are the truths[6] in the enacted statement. With these properties,

---

[6] In practice, it may be desirable to decouple truths from effects. For example, restricting effects to subset of *constant* truths ($effect(a, f) \triangleq \forall p : true(p \cup enacts(a), f)$) lets agents reason about the effects of each action via the possible justifications.

the framework structures the interaction between agents, and ensures that agents maintain a common understanding of actions' permission and effects.

The relaxation to agreement m_a permits Bob to take action a, which enacts m_b where $payload(m_b) = \{r_d\}$, and has agreement m_a and justification $\{m_a, m_b\}$. Amy and Bob agree that the action is permitted, because it satisfies Properties 3 to 6. Amy and Bob agree this action has one effect: Bob deletes Data1.

4   Case Study: Processing Distributed Medical Data

This section demonstrates an instantiation of the framework through an example application, using Datalog$^\neg$ as the policy language. Bas We adopt the example instantiation of Section 3, including the definitions of *agent*, *fact*, *extract*, *valid*, and so on. However, we instead instantiate *policies* as the syntactic category of all Datalog$^\neg$ programs. For example, Amy can author a statement with payload policy `ctl-knows(bob,amy)`, but the extracted policy is `ctl-knows(bob,amy). error`, which is invalid, as it has truths $\{$`ctl-knows(bob,amy)`, `error`$\}$. Furthermore, the statements of the *consortium* agent represents the consensus of the agents at large. Precisely, we instantiate *agreements* such that some agreement s is agreed for n if and only if s is the consortium's nth statement. Agents synchronise agreements, but only act on the most recent agreement.

Step 1 – We begin with a runtime session formalising a simple requirement: agents agree that it is an error for a data scientists to access data without the authorisation of the *administrator*. Initially, the only agreement contains s_1:

```
% Statement 's1' by 'consortium' (agreement at time 1)
owns(administrator, Data) :- ctl-accesses(Accessor, Data).
error :- ctl-accesses(Accessor, Data), owns(Owner, Data),
         not ctl-authorises(Owner, Accessor, Data).
```

Via a secondary communication channel, not reflected in the framework, agent *Amy* convinces the administrator to authorise Amy's access to data `x-rays`.

```
% Statement 's2' by 'administrator'
ctl-authorises(administrator, amy, x-rays).
```

Amy collects the above statements, and then states s_3 (below). Amy takes action a, which enacts s_3, and has basis s_1 and has justification $j \triangleq \{s_1, s_2, s_3\}$. For clarity, just this once, we detail the reasoning of an arbitrary observer of a given action; they draw the following conclusions about a:

- a preserves Property 3, as $j \subseteq$ *statements* (in fact, currently, $j =$ *statements*);
- a preserves Property 4, as $s_1 = basis(a)$, $s_3 = enacts(a)$, and $\{s_1, s_3\} \subseteq j$;
- a preserves Property 5, as where $p \triangleq extract(j)$, no p rule has the wrong author, so extraction preserves the payload-rules and adds no `error`, and fact `error` is not true in the resulting Datalog$^\neg$ model making p valid;
- a preserves Property 6, as a's basis applies at the time a is taken.

All observers agree that this action is permitted. Furthermore, they agree that a has the singleton set of effects $\{$`ctl-accesses(amy, x-rays)`$\}$, the truths of s_3. Amy mirrors this effect in the world outside, accessing the data identified by `x-rays`.

```
% Statement 's3' authored by 'amy'
ctl-accesses(amy, x-rays).
```

Anton "the antagonist" attempts to interfere with the normal operation of the system by making statements s_4 and s_5. However, no other agent takes any notice. Firstly, the policy extracted from s_4 is invalid, making it useless in justifying actions. Secondly, Statement s_5 is valid, but agents have no incentive to consider it, as it does not falsify `owns(administrator, x-rays)`, instead only further restricting access. Unlike agreement s_1, agents have no need to include s_5 in justifications.

```
% Statement 's4' authored by 'anton'
ctl-authorises(administrator, anton, x-rays).
```

```
% Statement 's5' authored by 'anton'
owns(anton, x-rays).
```

Step 2 – The administrator makes a new statement that empowers other agents to create authorisations (delegation). The administrator automatically authorises Bob's access to `x-rays` when authorized by both hospitals h_1 and h_2.

```
% Statement 's6' authored by 'administrator'
ctl-authorises(administrator, bob, x-rays) :- ctl-authorises(h1, bob, x-rays),
                                               ctl-authorises(h2, bob, x-rays).
```

Now, it suffices for Bob to acquire authorisation to access `x-rays` from the administrator or from (h_1 and h_2). Hospital h_1 further shares their power with h_2.

```
% Statement 's7' authored by 'h1'
ctl-authorises(h1, Accessor, x-rays) :- ctl-authorises(h2, Accessor, x-rays).
```

Now, it suffices for Bob to acquire authorisation to access `x-rays` from h_2 or from the administrator. Hospital h_2 grants this for all agents except Anton. Precisely, the authorisations of hospital h_2 are only applicable on the condition that `ctl-accesses(anton, x-rays)` is false. Thus, s_8 is useless to Anton.

```
% Statement 's8' authored by 'h2'
ctl-authorises(h2, Accessor, x-rays) :- ctl-accesses(Accessor, x-rays),
                                     not ctl-accesses(anton    , x-rays).
```

Bob states and enacts s_9 with justification $\{s_1, s_6, s_7, s_8, s_9\}$. Agents agree that this action is permitted and has the effect `ctl-accesses(bob, x-rays)`.

```
% Statement 's9' authored by 'bob'
ctl-accesses(bob, x-rays).
```

An (ex-ante) access control mechanism can prevent access to Anton by observing that there is no justification for the action with the effect `ctl-accesses(anton, x-rays)`. Alternatively, a monitoring or auditing mechanism may feed any observed access violation and message history to an ex-post enforcement authority.

Step 3 – The members of the consortium re-negotiate their agreement, because the administrator currently holds too much power. For example, the administrator stating `ctl-authorises(administrator, anton, x-rays)` would have sufficed to let Anton justify accessing `x-rays`, circumventing the need for authorisations from the hospitals. The new agreement s_{10} gives the power of authorisation to the (virtual)

consortium agent itself and distributes the power between the administrator and the hospitals: the hospitals label datasets as relevant for research into the flu (the illness) and the administrator determines which researcher can study the flu.

```
% Statement 's10' authored by 'consortium' (agreement at time 2)
error :- ctl-accesses(Accessor, Data),
    not ctl-authorises(consortium, Accessor, Data).
ctl-authorises(consortium, Accessor, Data)) :-
                    ctl-labels(administrator, Accessor, flu-researcher),
                    ctl-labels(Hospital     , Data     , flu-data),
                    ctl-hospital(consortium, Hospital).
ctl-hospital(consortium, h1).
ctl-hospital(consortium, h2).
```

Dan wishes to access `x-rays`. Dan petitions the administrator and hospital h_1 to make the necessary statements. The two agents cooperate.

```
% Statement 's11' authored by 'administrator'
ctl-labels(administrator, dan, flu-researcher).
```

```
% Statement 's12' authored by 'h1'
ctl-labels(h1, x-rays, flu-data).
```

Dan states and enacts s_{13} with justification $\{s_{10}, s_{11}, s_{12}, s_{13}\}$. Agents agree that this action is permitted and has the effect `ctl-accesses(dan, x-rays)`.

```
% Statement 's13' authored by 'dan'
ctl-accesses(dan, x-rays).
```

Step 4 – We demonstrate how agents can act in complete confidence that their action is permitted, despite partial knowledge of the existing statements.

Hospital h_1 automates their labelling of flu data based on the consent of data subjects. However, consent is itself sensitive information; s_{14} would let an observer infer that the scans are connected to Caterina, a hospital patient. As such, hospital h_1 keeps s_{14} private; the statement is not gossiped to other agents.

```
% Statement 's14' authored by 'h1'
ctl-labels(h1, cat-scans, flu-data) :- ctl-consent(caterina, cat-scans).
```

Dan petitions hospital h_1 to label `cat-scans` as flu data, such that Dan can access it. The hospital does not grant Dan access immediately, but instead, acquires statement s_{15} (below) from Caterina, and then sends s_{14} and s_{15} to Dan, with the understanding that Dan will keep these statements private.

```
% Statement 's15' authored by 'caterina'
ctl-consent(caterina, cat-scans).
```

Dan states and enacts s_{16} with justification $\{s_{10}, s_{11}, s_{14}, s_{15}, s_{16}\}$. Dan publicises the statement s_{16}, but keeps the justification private. The system is established with rules for access to justifications with sensitive information. The justification is checked only by auditors with special permissions. For example, an automated auditor runs in a contained environment that leaks no information and only publicises the answer to the question: is the given action justified?

```
% Statement 's16' authored by 'dan'
ctl-accesses(dan, cat-scans).
```

Other agents negotiate and enact data accesses despite ignorance of the sensitive statements s_{14} and s_{15}. Their own actions and permissions are unaffected.

5   Implementation

This section discusses the considerations for implementing our framework in general, and describes our ongoing prototype implementation in particular.

5.1   Implementation of the Framework Statics

Section 3.1 specifies the requirements on policy language needed to instantiate the framework. Here, we briefly consider the practical design space in general, and describe the approach of our prototype implementation.

Deterministic Truth To be instantiated, the framework must define *truth* as a static relation over policies and *facts*. This implies a *deterministic* semantics: the same policies always denote the same truths. This ensures that the communication of policies realises a consistent communication of understanding about the domain of discourse. For example, (complete) Prolog is unsuitable as a policy language as its inference procedure is non-terminating. Even if all agents agree on a Prolog program, generally, no agent can be certain that the conclusions drawn will be replicated consistently, e.g., by an auditor checking the validity of a key policy.

Our prototype policy language uses the operational form [39] of the well-founded semantics negation [56], assigning a unique value to each fact by isolating logical contradictions to *unknown values*. For example, p is unknown in `p :- not p`. This semantics confers the necessary property: inference of each program always terminates, and each truth is replicated by every agent every time.

Complex Facts and Reflective Extraction Section 3.1 and Section 4 demonstrated two (similar) definitions of the *extract* function, demonstrating its role in controlling which policies authors can express.

Our prototype generalises Datalog¬ by supporting the construction of *complex facts* like `knows(amy,owns(bob,x-rays))` from other facts. This feature affords a powerful definition of *extract* that more completely reflects the relation between true facts, statements, and authors. Precisely, *extract* injects rules into the policy that reflect the message context at a finer granularity, affording agents more context-sensitive control. For example, agent a cannot assert the truth of fact f with a rule in message m without also asserting the truth of facts `authors`(a, m) and `has-rule-asserting`(m, a). Thus, policies can reflect on the system dynamics, giving their authors more fine-grained control over actions. For example, statements can explicitly refer to particular statements, and condition validity on the author of rules resulting in the truth of particular facts.

5.2   Implementation of the Framework Dynamics

Our prototype implements a proposed policy-enforcement component [17] for the EPI framework [27]. This prescribes the agents and actions relevant to the use case, similar to our demonstration in Section 4. The agents include

processors, administrators, and owners of sensitive data (hospitals and their patients). The agreement formalises the dependency of processors on the owners. Firstly, with the assistance of planner agents, processors make statements defining data-processing workflows. Secondly, the relevant data owners make statements authorising workflows, permitting their data access and processing actions.

Administrative agents are trusted to mediate the collection and transport of statements between data owners and processors. Agents communicate statements asynchronously, and only synchronise periodically to check for changes to the agreement. The agreement is changed only by the administrators in two circumstances: members join or leave the consortium, or to temporarily halt processing while the infrastructure underlying workflow processing is maintained.

6   Discussion

We discuss the strengths and limitations of (systems instantiating) the framework defined in section 3, and demonstrated in Section 4.

6.1   Strengths

Highly Dynamic and Extensible The framework is highly abstract, making it applicable to many policy languages and runtime systems.

Notably, the framework is parametric to the statically-defined policy language, admitting any formal language affording pure functions of truth and validity. This leaves significant room for systems instantiating the framework to adopt various notions of policy, with various information models, and various semantics. For example, the framework's notion of policy affords the n-ary relations and logical constraints underlying Bell-LaPadua security policies and various Rule-Based Access control policies, which are all summarised and compared in [60]. However, we leave the precise instantiation of these kinds of policy to future work.

Moreover, the relations between agents that ultimately control actions are highly dynamically configurable, by agents making statements. Different configurations confer different characteristics on the system, for example, allowing for dynamic specialisation for various use cases in reaction to runtime information. We recognise two noteworthy spectra on which particular system configurations fall. Together, these help to clarify the ways systems can change their characteristics at runtime. Firstly, systems can be centralised (where inter-agent consistency is high) or decentralised (where many agents have meaningful power to act). Secondly, systems can be highly static (where properties of interest are preserved) or dynamic (where properties can be changed by agents at runtime).

Formal Agent Power Dynamics Agents are specified to remain well-behaved (Definition 1): agents take only permitted actions. Well-behavedness enables complex power dynamics, because action permission is influenced by statements and agreements. This lets systems model various common and useful normative

concepts by delegating power. For example, Section 4 demonstrates the administrator sharing power with the hospitals, and also the partitioning of consortium power over the administrator and the hospitals.

Well-behavedness is robust; an agent that violates their own well-behavedness by taking a non-permitted action preserves the well-behavedness of their peers. The permission of actions is independent of (the permission) of actions; each agent must independently justify their actions from the available statements.

Autonomy and Parallelism Agents synchronise to change agreements. All other communication can be asynchronous, delayed, and lossy. Agents are very autonomous, as they are never fundamentally compelled to act or make statements. Hence, agents are robust to unreliable peers. The framework affords realistic inter-agent enforcement of well-behavedness: agents monitor actions, and actors bear the burden of proving that their actions are permitted. Thus, agents must retain their actions' justifications. However, other statements and actions can be forgotten (e.g., to free memory) while preserving the framework guarantees.

Consistent Permission despite Privacy Agents always agree which actions are permitted, despite being defined by statements not known to all agents. This apparent contradiction is resolved by agents being able to decide when their known statements suffice for permission. Moreover, permission is objective; agents are certain that other agents (e.g., future auditors) agree that their actions were permitted, without involving them at all. For example, in Section 4, Dan observes s_{15} and Amy does not, but both agree that both are well-behaved.

6.2   Limitations

Costly Justification Search To remain well-behaved, agents must search for justifications that permit their desired actions. In general, this problem is undecidable. In practice, the difficulty depends on which statements are available, and on the characteristics of the policy language. For example, the operational semantics of Datalog¬ is desirably tractable. Moreover, this search problem is reducible to many problems well-studied in the literature. We are particularly interested in applying answer set solving (e.g., with Clingo [21]) or model-checking rewrite systems modulo theories (e.g., with Maude [13]) to this problem in future. The hard, general problem is made tractable by restricting policies.

No Obligation to Act (in Time) Our framework offers no fundamental mechanism for agents to compel one another to action. Thus, the framework cannot internalise normative *obligations to act*. These features can be added to systems implementing our framework, but they may have difficulty in compelling agents to act before specific deadlines, as a consequence of the aforementioned point on the search for justifications. For this reason, our framework is not a natural choice for implementations of *real-time systems*, where agents must react

to stimuli within strict time limits. Future work can investigate constraints on system configurations that strike desirable compromises between system flexibility on the one hand, and predictability of actions (and their timing) on the other.

Specification of Communication As specified in Section 3.2, and demonstrated in Section 4, agents can decide that their actions are universally permitted, despite each having only partial knowledge of the existing statements and actions. However, the framework itself does not prescribe how messages and actions are shared between agents. In future work, we want to supplement the current *(justification) agreements* with *sharing agreements*, which specify how agents share (e.g., gossip) their statements and actions with their peers.

No Privacy from the Actor Agents are only able to act on permissions defined by *known* statements. For example, in Section 4, for Amy to justify access, Amy acquires the patient's consent, revealing the patient identity to Amy. This can be worked around; trusted intermediaries can forward transformed policies to hide private information. In this case, the hospital can authorise Amy's access (unconditionally) after observing the patient consent themselves. In future, we want to systematise these intermediary transformations such that the hidden policy information can always be recovered (e.g., by trusted auditors).

7   Related work

Our framework prescribes a relation between concerns that are each independently explored in the literature: 1. blockchains can synchronise dynamic, decentralised policies, 2. trust management specifies the delegation of power between agents, and 3. Curie evaluates data access policies as a function of sensitive data.

Smart Ledgers atop Blockchains Distributed ledgers provide a (probabilistic) means of consensus on the state between decentralised processes. Blockchains are a technology for implementing distributed ledgers. For example, Fabric emphasises scalability [4], while Ouroboros emphasises provable security [31].

Systems such as SmartAccess [37] use distributed ledgers to store policies and (meta-)data, enabling decentralised implementations of the access-control model. However, these technologies synchronise policies, making them unsuitable for private policy information (e.g., the consent rule in s_{16} in Section 4).

Other ledger-based systems allow a heterogeneous view on the policy state. For example, Canton [15] (whitepaper) replaces the (sequential) block*chain* with a (hierarchical) block*tree*. Agents must only synchronise the relevant sub-trees with their neighbours. This lays the groundwork for private policies.

We see ledgers and blockchains as one way for our agents to synchronise agent agreements in particular. However, they are unsuitable for distributing our agent statements in general, as their ordering and synchronisation is unnecessary.

Curie Curie is a policy-based data exchange system [11]. Our works share a fundamentally decentralised approach to the specification and enforcement of formal policies to regulate the exchange of data, based on consortium agreements and local policies. Moreover, both works define permission to act in terms of assembled policies, for Curie, of dual policies of x sharing with y and y acquiring from x. We also see similarities between Curie and the EPI framework in their shared application to federated machine learning with sensitive medical data.

A significant feature of Curie is its specification and evaluation of data sharing policies as a function of the shared data itself. Policy decisions digest the homomorphically encrypted data [16], revealing only the evaluation result in plain text. For example, a hospital's sharing policy is conditioned on the shared data surpassing a threshold of differential privacy [35].

Our work abstracts away from the relation between policies and the effects of actions (e.g., sharing data). Instead, we focus on relating policies and agent actions and communications. Our approach affords a fundamentally multi-party approach to policy decisions. Permissions arise from the composition of statements from different agents at different times. Thus, our statements more extensively internalise the negotiation and refinement of multi-party power dynamics. For example, Section 4 demonstrates multiple agents participating in the incremental sharing of power to permit data access requests. By decoupling statements from their authors, statements meaningfully delegate reasoning and action to other agents. For example, also in Section 4, the consortium agent begins with sole power over authorisation, but after the agreement, it plays no further part.

As such, our contributions are largely orthogonal to Curie, and we are inspired to investigate the combination of the works' best features. Can (data sharing) actions be justified as a function of the shared data itself?

Trust Management Traditional access control develops languages and tools for specifying and checking a requester's permission to access data. *Trust management* reifies the role of the accessor as a *certificate*, primarily, to enable access control in a decentralised environment, where the identities of particular requesters are not known ahead of time [9]. Much literature dates to the 1980's and 1990's, investigating policy languages suited to defining certificates and inferring them at request time from context. Many of these are specialised extensions of Datalog, adding non-monotonicity [32], constraints [33], and weights [8].

Like access control, trust management focuses agent reasoning on the access-request decision, whereas our framework emphasises the inter-relationship between agents and their actions via their synchronised agreements. However, the bulk of trust management research complements our work, because it informs the selection of particular policy languages suited to particular purposes. [43] overviews and compares (the complexity of) noteworthy trust management languages.

8   Conclusion

We define a framework for policy-driven data exchange, which minimises the requirements on the policy language itself, maximising applicability to existing work. The framework focuses on defining the relation between agent statements and the permissions on agent actions. Ultimately, we show that any definition of framework objects that satisfies our realistic requirements satisfies a useful property. Namely, agents can decide which actions are permitted, such that all other agents certainly agree. Moreover, agents are confident their decisions are universal, e.g., shared by all peers, including monitors and auditors.

Our work is motivated by its application to the exchange of sensitive (e.g., medical) data between autonomous agents, where the policies themselves are potentially private. Crucially, agents can decide, on a case-by-case basis, with whom they share their dynamic policy statements, balancing policy privacy on the one hand, with permitting peers' actions on the other hand.

Work continues to develop a policy component in the EPI framework, an existing, federated, medical workflow processing system. This entails developing our own specialised policy language, and implementing the agent reasoning and communications, by drawing from several related works.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Akaichi, I., Kirrane, S.: Usage control specification, enforcement, and robustness: A survey. CoRR **abs/2203.04800** (2022). `https://doi.org/10.48550/arXiv.2203.04800`

2. Anderson, A., Nadalin, A., Parducci, B., Engovatov, D., Lockhart, H., Kudo, M., Humenn, P., Godik, S., Anderson, S., Crocker, S., et al.: extensible access control markup language (xacml) version 1.0. Oasis (2003)

3. Andrighetto, G., Governatori, G., Noriega, P., van der Torre, L.W.N. (eds.): Normative Multi-Agent Systems, Dagstuhl Follow-Ups, vol. 4. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2013)

4. Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., Caro, A.D., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., Muralidharan, S., Murthy, C., Nguyen, B., Sethi, M., Singh, G., Smith, K., Sorniotti, A., Stathakopoulou, C., Vukolic, M., Cocco, S.W., Yellick, J.: Hyperledger fabric: a distributed operating system for permissioned blockchains. In: Oliveira, R., Felber, P., Hu, Y.C. (eds.) Proceedings of the Thirteenth EuroSys Conference, EuroSys 2018, Porto, Portugal, April 23-26, 2018. pp. 30:1–30:15. ACM (2018). `https://doi.org/10.1145/3190508.3190538`

5. Bakhshi, R., Cloth, L., Fokkink, W.J., Haverkort, B.R.: Meanfield analysis for the evaluation of gossip protocols. SIGMETRICS Perform. Evaluation Rev. **36**(3), 31–39 (2008). `https://doi.org/10.1145/1481506.1481513`

6. van Binsbergen, L.T., Kebede, M.G., Baugh, J., van Engers, T.M., van Vuurden, D.G.: Dynamic generation of access control policies from social policies. In: Varandas, N., Yasar, A., Malik, H., Galland, S. (eds.) The 12th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN 2021) / The 11th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH-2021), Leuven, Belgium, November 1-4, 2021. Procedia Computer Science, vol. 198, pp. 140–147. Elsevier (2021). `https://doi.org/10.1016/j.procs.2021.12.221`

7. van Binsbergen, L.T., Liu, L., van Doesburg, R., van Engers, T.M.: eflint: a domain-specific language for executable norm specifications. In: Erwig, M., Gray, J. (eds.) GPCE '20: Proceedings of the 19th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences, Virtual Event, USA, November 16-17, 2020. pp. 124–136. ACM (2020). `https://doi.org/10.1145/3425898.3426958`

8. Bistarelli, S., Martinelli, F., Santini, F.: Weighted datalog and levels of trust. In: Proceedings of the The Third International Conference on Availability, Reliability and Security, ARES 2008, March 4-7, 2008, Technical University of Catalonia, Barcelona , Spain. pp. 1128–1134. IEEE Computer Society (2008). `https://doi.org/10.1109/ARES.2008.197`

9. Blaze, M., Feigenbaum, J., Lacy, J.: Decentralized trust management. In: 1996 IEEE Symposium on Security and Privacy, May 6-8, 1996, Oakland, CA, USA. pp. 164–173. IEEE Computer Society (1996). `https://doi.org/10.1109/SECPRI.1996.502679`

10. Boella, G., van der Torre, L.W.N.: Substantive and procedural norms in normative multiagent systems. J. Appl. Log. **6**(2), 152–171 (2008). `https://doi.org/10.1016/j.jal.2007.06.006`

11. Celik, Z.B., Acar, A., Aksu, H., Sheatsley, R., McDaniel, P.D., Uluagac, A.S.: Curie: Policy-based secure data exchange. In: Ahn, G., Thuraisingham, B., Kantarcioglu, M., Krishnan, R. (eds.) Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy, CODASPY 2019, Richardson, TX, USA, March 25-27, 2019. pp. 121–132. ACM (2019). `https://doi.org/10.1145/3292006.3300042`

12. Ceri, S., Gottlob, G., Tanca, L.: What you always wanted to know about datalog (and never dared to ask). IEEE Trans. Knowl. Data Eng. **1**(1), 146–166 (1989). `https://doi.org/10.1109/69.43410`

13. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.L.: The maude 2.0 system. In: Nieuwenhuis, R. (ed.) Rewriting Techniques and Applications, 14th International Conference, RTA 2003, Valencia, Spain, June 9-11, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2706, pp. 76–87. Springer (2003). `https://doi.org/10.1007/3-540-44881-0_7`

14. Curry, E., Tuikka, T.: An organizational maturity model fordata spaces: A data sharing wheel approach. In: Curry, E., Scerri, S., Tuikka, T. (eds.) Data Spaces - Design, Deployment and Future Directions, pp. 21–42. Springer (2022). `https://doi.org/10.1007/978-3-030-98636-0_2`

15. Digital Asset: Canton network: A network of networks for smart contract applications. Available at `https://www.digitalasset.com/hubfs/Canton/Canton%20Network%20-%20White%20Paper.pdf` (Accessed: 23-2-2024), (Whitepaper)

16. Doan, T.V.T., Messai, M., Gavin, G., Darmont, J.: A survey on implementations of homomorphic encryption schemes. J. Supercomput. **79**(13), 15098–15139 (2023). `https://doi.org/10.1007/S11227-023-05233-Z`

17. Esterhuyse, C.A., Müller, T., van Binsbergen, L.T., Belloum, A.S.Z.: Exploring the enforcement of private, dynamic policies on medical workflow execution. In: 18th IEEE International Conference on e-Science, e-Science 2022, Salt Lake City, UT, USA, October 11-14, 2022. pp. 481–486. IEEE (2022). `https://doi.org/10.1109/ESCIENCE55777.2022.00086`

18. European Commission: Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) (Text with EEA relevance) (2016), `https://eur-lex.europa.eu/eli/reg/2016/679/oj`

19. Fernandez, R.C.: Data-sharing markets: Model, protocol, and algorithms to incentivize the formation of data-sharing consortia. Proc. ACM Manag. Data **1**(2), 172:1–172:25 (2023). `https://doi.org/10.1145/3589317`

20. Fragkos, G., Johnson, J., Tsiropoulou, E.: Dynamic role-based access control policy for smart grid applications: An offline deep reinforcement learning approach. IEEE Trans. Hum. Mach. Syst. **52**(4), 761–773 (2022). `https://doi.org/10.1109/THMS.2022.3163185`

21. Gebser, M., Kaufmann, B., Kaminski, R., Ostrowski, M., Schaub, T., Schneider, M.: Potassco: The potsdam answer set solving collection. AI Commun. **24**(2), 107–124 (2011). `https://doi.org/10.3233/AIC-2011-0491`

22. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Kowalski, R.A., Bowen, K.A. (eds.) Logic Programming, Proceedings of the Fifth International Conference and Symposium, Seattle, Washington, USA, August 15-19, 1988 (2 Volumes). pp. 1070–1080. MIT Press (1988)

23. Governatori, G., Idelberger, F., Milosevic, Z., Riveret, R., Sartor, G., Xu, X.: On legal contracts, imperative and declarative smart contracts, and blockchain systems. Artif. Intell. Law **26**(4), 377–409 (2018). `https://doi.org/10.1007/s10506-018-9223-3`

24. Ianella, R.: Open digital rights language (odrl). Open Content Licensing: Cultivating the Creative Commons (2007)

25. Jung, C., Dörr, J.: Data usage control. In: Otto, B., ten Hompel, M., Wrobel, S. (eds.) Designing Data Spaces: The Ecosystem Approach to Competitive Advantage, pp. 129–146. Springer (2022). `https://doi.org/10.1007/978-3-030-93975-5_8`

26. Kassem, J.A., de Laat, C., Taal, A., Grosso, P.: The EPI framework: A dynamic data sharing framework for healthcare use cases. IEEE Access **8**, 179909–179920 (2020). `https://doi.org/10.1109/ACCESS.2020.3028051`

27. Kassem, J.A., Valkering, O., Belloum, A., Grosso, P.: EPI framework: Approach for traffic redirection through containerised network functions. In: 17th IEEE International Conference on eScience, eScience 2021, Innsbruck, Austria, September 20-23, 2021. pp. 80–89. IEEE (2021). `https://doi.org/10.1109/eScience51609.2021.00018`

28. Ketsman, B., Koch, C.: Datalog with negation and monotonicity. In: Lutz, C., Jung, J.C. (eds.) 23rd International Conference on Database Theory, ICDT 2020, March 30-April 2, 2020, Copenhagen, Denmark. LIPIcs, vol. 155, pp. 19:1–19:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020). `https://doi.org/10.4230/LIPIcs.ICDT.2020.19`

29. Khan, M., den Hartog, F.T.H., Hu, J.: A survey and ontology of blockchain consensus algorithms for resource-constrained iot systems. Sensors **22**(21), 8188 (2022). `https://doi.org/10.3390/S22218188`

30. Khobragade, P., Turuk, A.K.: Blockchain consensus algorithms: A survey. In: Prieto, J., Martínez, F.L.B., Ferretti, S., Guardeño, D.A., Nevado-Batalla, P.T. (eds.) Blockchain and Applications, 4th International Congress, BLOCKCHAIN 2022, L'Aquila, Italy, 13-15 July 2022. Lecture Notes in Networks and Systems, vol. 595, pp. 198–210. Springer (2022). `https://doi.org/10.1007/978-3-031-21229-1_19`

31. Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: A provably secure proof-of-stake blockchain protocol. In: Katz, J., Shacham, H. (eds.) Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I. Lecture Notes in Computer Science, vol. 10401, pp. 357–388. Springer (2017). `https://doi.org/10.1007/978-3-319-63688-7_12`

32. Li, N., Grosof, B.N., Feigenbaum, J.: Delegation logic: A logic-based approach to distributed authorization. ACM Trans. Inf. Syst. Secur. **6**(1), 128–171 (2003). `https://doi.org/10.1145/605434.605438`

33. Li, N., Mitchell, J.C.: DATALOG with constraints: A foundation for trust management languages. In: Dahl, V., Wadler, P. (eds.) Practical Aspects of Declarative Languages, 5th International Symposium, PADL 2003, New Orleans, LA, USA, January 13-14, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2562, pp. 58–73. Springer (2003). `https://doi.org/10.1007/3-540-36388-2_6`

34. Mao, X., Wang, Q., Yang, S.: A survey of agent-oriented programming from software engineering perspective. Web Intell. **15**(2), 143–163 (2017). `https://doi.org/10.3233/WEB-170357`

35. Mohassel, P., Zhang, Y.: Secureml: A system for scalable privacy-preserving machine learning. In: 2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017. pp. 19–38. IEEE Computer Society (2017). `https://doi.org/10.1109/SP.2017.12`

36. Munoz-Arcentales, A., López-Pernas, S., Pozo, A., Alonso, Á., Salvachúa, J., Huecas, G.: An architecture for providing data usage and access control in data sharing ecosystems. In: Shakshuki, E.M., Yasar, A., Malik, H. (eds.) The 10th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN 2019) / The 9th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH-2019) / Affiliated Workshops, Coimbra, Portugal, November 4-7, 2019. Procedia Computer Science, vol. 160, pp. 590–597. Elsevier (2019). `https://doi.org/10.1016/J.PROCS.2019.11.042`

37. de Oliveira, M.T., Reis, L.H.A., Verginadis, Y., Mattos, D.M.F., Olabarriaga, S.D.: Smartaccess: Attribute-based access control system for medical records based on smart contracts. IEEE Access **10**, 117836–117854 (2022). `https://doi.org/10.1109/ACCESS.2022.3217201`

38. Parvizimosaed, A., Roveri, M., Rasti, A., Amyot, D., Logrippo, L., Mylopoulos, J.: Model-checking legal contracts with symboleopc. In: Syriani, E., Sahraoui, H.A., Bencomo, N., Wimmer, M. (eds.) Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems, MODELS 2022, Montreal, Quebec, Canada, October 23-28, 2022. pp. 278–288. ACM (2022). `https://doi.org/10.1145/3550355.3552449`

39. Przymusinski, T.C.: The well-founded semantics coincides with the three-valued stable semantics. Fundam. Inform. **13**(4), 445–463 (1990)

40. Qiu, J., Tian, Z., Du, C., Zuo, Q., Su, S., Fang, B.: A survey on access control in the age of internet of things. IEEE Internet Things J. **7**(6), 4682–4696 (2020). `https://doi.org/10.1109/JIOT.2020.2969326`

41. Ren, W., Beard, R.W., Atkins, E.M.: A survey of consensus problems in multi-agent coordination. In: American Control Conference, ACC 2005, Portland, OR, USA, 8-10 June, 2005. pp. 1859–1864. IEEE (2005). `https://doi.org/10.1109/ACC.2005.1470239`

42. Ross, K.A.: Modular stratification and magic sets for DATALOG programs with negation. In: Rosenkrantz, D.J., Sagiv, Y. (eds.) Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, April 2-4, 1990, Nashville, Tennessee, USA. pp. 161–171. ACM Press (1990). `https://doi.org/10.1145/298514.298558`

43. Sacha, K.: Trust management languages and complexity. In: Meersman, R., Dillon, T.S., Herrero, P., Kumar, A., Reichert, M., Qing, L., Ooi, B.C., Damiani, E., Schmidt, D.C., White, J., Hauswirth, M., Hitzler, P., Mohania, M.K. (eds.) On the Move to Meaningful Internet Systems: OTM 2011 - Confederated International Conferences: CoopIS, DOA-SVI, and ODBASE 2011, Hersonissos, Crete, Greece, October 17-21, 2011, Proceedings, Part II. Lecture Notes in Computer Science, vol. 7045, pp. 588–604. Springer (2011). `https://doi.org/10.1007/978-3-642-25106-1_12`

44. Samarati, P., di Vimercati, S.D.C.: Access control: Policies, models, and mechanisms. In: Focardi, R., Gorrieri, R. (eds.) Foundations of Security Analysis and Design, Tutorial Lectures [revised versions of lectures given during the IFIP WG 1.7 International School on Foundations of Security Analysis and Design, FOSAD 2000, Bertinoro, Italy, September 2000]. Lecture Notes in Computer Science, vol. 2171, pp. 137–196. Springer (2000). `https://doi.org/10.1007/3-540-45608-2_3`

45. Sandhu, R.S.: Role-based access control. Adv. Comput. **46**, 237–286 (1998). `https://doi.org/10.1016/S0065-2458(08)60206-5`

46. Schmidt, D.C., et al.: Model-driven engineering. Computer-IEEE Computer Society-**39**(2), 25 (2006)

47. Servos, D., Osborn, S.L.: Current research and open problems in attribute-based access control. ACM Comput. Surv. **49**(4), 65:1–65:45 (2017). `https://doi.org/10.1145/3007204`

48. Shakeri, S., Maccatrozzo, V., Veen, L.E., Bakhshi, R., Gommans, L., de Laat, C., Grosso, P.: Modeling and matching digital data marketplace policies. In: 15th International Conference on eScience, eScience 2019, San Diego, CA, USA, September 24-27, 2019. pp. 570–577. IEEE (2019). `https://doi.org/10.1109/ESCIENCE.2019.00078`

49. Sharifi, S., Parvizimosaed, A., Amyot, D., Logrippo, L., Mylopoulos, J.: Symboleo: Towards a specification language for legal contracts. In: Breaux, T.D., Zisman, A., Fricker, S., Glinz, M. (eds.) 28th IEEE International Requirements Engineering Conference, RE 2020, Zurich, Switzerland, August 31 - September 4, 2020. pp. 364–369. IEEE (2020). `https://doi.org/10.1109/RE48521.2020.00049`

50. Shi, B., Zhou, A.: Bottom-up evaluation of datalog with negation. J. Comput. Sci. Technol. **9**(3), 229–244 (1994). `https://doi.org/10.1007/BF02939504`

51. Shoham, Y.: Agent-oriented programming. Artif. Intell. **60**(1), 51–92 (1993). `https://doi.org/10.1016/0004-3702(93)90034-9`

52. Strasser, C., Antonelli, G.A.: Non-monotonic Logic. In: Zalta, E.N. (ed.) The Stanford Encyclopedia of Philosophy. Metaphysics Research Lab, Stanford University, Summer 2019 edn. (2019)

53. Torre-Bastida, A.I., Gil, G., Miñón, R., Díaz-de-Arcaya, J.: Technological perspective of data governance in data space ecosystems. In: Curry, E., Scerri, S., Tuikka, T. (eds.) Data Spaces - Design, Deployment and Future Directions, pp. 65–87. Springer (2022). `https://doi.org/10.1007/978-3-030-98636-0_4`

54. Um-e-Ghazia, Masood, R., Shibli, M.A., Bilal, M.: Usage control model specification in XACML policy language - XACML policy engine of UCON. In: Cortesi, A., Chaki, N., Saeed, K., Wierzchon, S.T. (eds.) Computer Information Systems and Industrial Management - 11th IFIP TC 8 International Conference, CISIM 2012, Venice, Italy, September 26-28, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7564, pp. 68–79. Springer (2012). https://doi.org/10.1007/978-3-642-33260-9_5

55. Valkering, O., Cushing, R., Belloum, A.: Brane: A framework for programmable orchestration of multi-site applications. In: 17th IEEE International Conference on eScience, eScience 2021, Innsbruck, Austria, September 20-23, 2021. pp. 277–282. IEEE (2021). https://doi.org/10.1109/ESCIENCE51609.2021.00056

56. Van Gelder, A., Ross, K.A., Schlipf, J.S.: The well-founded semantics for general logic programs. Journal of the ACM **38**(3), 619–649 (Jul 1991). https://doi.org/10.1145/116825.116838

57. Viganò, F., Colombetti, M.: Symbolic model checking of institutions. In: Gini, M.L., Kauffman, R.J., Sarppo, D., Dellarocas, C., Dignum, F. (eds.) Proceedings of the 9th International Conference on Electronic Commerce: The Wireless World of Electronic Commerce, 2007, University of Minnesota, Minneapolis, MN, USA, August 19-22, 2007. ACM International Conference Proceeding Series, vol. 258, pp. 35–44. ACM (2007). https://doi.org/10.1145/1282100.1282109

58. Wesley, N.H.: Some fundamental legal conceptions as applied in judicial reasoning. The Yale Law Journal **23**(1), 16 (1913)

59. Zhang, X., Parisi-Presicce, F., Sandhu, R.S., Park, J.: Formal model and policy specification of usage control. ACM Trans. Inf. Syst. Secur. **8**(4), 351–387 (2005). https://doi.org/10.1145/1108906.1108908

60. Zhao, G., Chadwick, D.W.: On the modeling of bell-lapadula security policies using RBAC. In: 17th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises, WETICE 2008, Rome, Italy, June 23-25, 2008, Proceedings. pp. 257–262. IEEE Computer Society (2008). https://doi.org/10.1109/WETICE.2008.34

61. Zhou, X., Belloum, A., Lees, M.H., van Engers, T.M., de Laat, C.: The dynamics of corruption under an optional external supervision service. Appl. Math. Comput. **457**, 128172 (2023). https://doi.org/10.1016/J.AMC.2023.128172