

THE RESEARCH UNIVERSITY PRESS

International
Computational Law Journal

VOLUME 2 / ISSUE 4 / CURATED SOURCE EDITION

# Building Net-Native Agreement Systems

Joshua Z. Tan · Luke V. Miller

MIT Computational Law Report · Originally published 2022-12-05

Independent reader edition under CC BY 4.0. This attribution cover was added; original source PDF pages follow unchanged. Original authorship and source identity are retained. MIT, Stanford University, Stanford Law School and CodeX have not sponsored or endorsed this republication.

Source, license and version evidence accompany the online article.

**MIT Computational Law Report**

# Building Net-Native Agreement Systems

**Joshua Z. Tan Luke V. Miller**

**Published on:** Dec 05, 2022

**URL:** https://law.mit.edu/pub/buildingnetnativeagreementsystems

**License:** Creative Commons Attribution 4.0 International License (CC-BY 4.0)

**ABSTRACT**

*Agreements and contracts are everywhere, but they are built on layers and layers of legal and social institutions. Software is slowly entering into this stack. In this article, we introduce agreement paths, a general model for understanding and decomposing digital agreement systems, and Agreement Engine, an open-source software service for building net-native agreement systems. We demonstrate Agreement Engine by building two example agreement systems: Scarce Knowledge, an app for crowdfunding essays, and Twitter Social Capital, a bot that allows users to form and enforce Twitter agreements.*

## Introduction

Agreements are the building blocks of modern societies. Whenever you want to work with other people—whether it's starting a business, building open-source software, or slaying a (virtual) dragon—you will need to first form an agreement with those people to communicate expectations, safeguard rights, or divvy up rewards. When we change the agreements that people can form, for example through policies that affect communication technology, social trust, or the rule of law, we change the kinds of economies and politics that people produce.

But what determines the agreements that people can form? Or, to put it another way: for a given kind of agreement (legal, informal, computational, etc.), how can we build *systems* for authoring and enforcing such agreements? Such agreement systems range widely: from a state's business registration process, to the small claims court system, to the Ethereum smart contract system, to the guild systems of online games like World of Warcraft or EVE Online, to informal systems of agreement that govern everything from bazaars to traffic circles.

In this article, we introduce agreement paths, a general formalism for modeling agreement systems, and Agreement Engine, a software service for building net-native agreement systems. In doing so, we emphasize (1) the growing importance of agreement systems that are implemented partially or wholly online and (2) the benefits of a more rigorous and interoperable framework for engineering such systems. We also present two experimental agreement systems, Scarce Knowledge and Twitter Social Capital, built using Agreement Engine. Finally, we contextualize our contributions within the larger project of building net-native agreement systems and close out with a discussion of the user experience of legal contracts compared to smart contracts or informal agreements.

## Background and related work

What do the following have in common?

1. A signed rental contract
2. A one-line email: "cool, meet you at 9pm outside the bar"

3. A (multi-party) smart contract
4. A bid on eBay
5. A video of two people exchanging nods

Answer: they are all manifestations of mutual assent, a.k.a. *agreements*. *Mutual assent* is just a piece of jargon that means “we agree”—it’s an abstract relation (what the law calls a ‘meeting of the minds’) between two or more subjects that share a common intention or belief. Agreements are *manifestations* of mutual assent, meaning that they bear witness to the claim that a common intention or belief exists.[1] To put it another way: every agreement constitutes evidence of mutual assent, though not every such agreement is thereby made enforceable.

Many agreements also incur *obligations*, where an obligation is an action that one has to do. These obligations usually assume certain mechanisms for *enforcement*, where *enforcement* is the act of compelling compliance with an obligation. For example: an email thread between two friends agreeing to meet at a bar (1) is explicit evidence of mutual assent and (2) creates a social obligation to meet at said bar where (3) the obligation to show up is enforced by an implied social norm: don’t break promises to friends or you’ll lose your friends and/or be reputed as a flake. A bid on eBay, on the other hand, (1) is evidence of the bidder’s agreement to buy the item at that price and (2) creates an obligation to pay the seller at that price if they win the auction where (3) that obligation is presumptively legally enforceable (4) either through eBay’s automated escrow system or through eBay’s (less-automated) unpaid item policy.[2]

This (1)-(2)-(3) pattern of agreement-obligation-enforcement is extremely common in society and in the law. We call this pattern a *contract system*, and we call agreements that fit this pattern *contracts*. There is substantial theoretical and empirical work on contract systems within comparative law[3], contract law[4], philosophy[5], and economics[6], mostly focusing on national legal contract systems. Social scientists have also studied informal agreement systems—informal only in the sense that they do not directly involve a court system—through an institutional lens.[7] And most recently, smart contract systems—systems of agreements automatically enforced through code—have matured from early conceptual models[8] into large, blockchain-based platforms such as Ethereum[9] and Cosmos[10].

In what follows, we will focus on a number of features shared between traditional legal contract systems, informal agreement systems, and smart contract systems.

| Term | Definition |
| --- | --- |
| Agreement | A manifestation of mutual assent between a number of parties |
| Obligation | An action that one has to do |

| | |
|---|---|
| Enforcement | The act of compelling compliance with an obligation |
| Contract | An agreement that creates an obligation with an enforcement mechanism |
| Contract system | A system for creating and enforcing contracts |
| Legal contract | An agreement recognized and enforced by the state, typically through a system of laws, courts, and enforcement agencies |
| Smart contract | An agreement written in code and enforced by a technical system such as a virtual world or blockchain |
| Informal agreement | An agreement not intended to be enforced through a legal or technical mechanism but instead through social norms, reputational incentives, and/or extralegal means that do not depend on a formal institutional enforcement mechanism |
| Joint account | A digital representation of an n-person agreement, akin to a user account |
| Registration | The process of recording or representing a completed agreement and related data, usually by a qualified authority |
| Authentication | The process of checking the data, status, and/or registration of an agreement, usually by an enforcement mechanism |

> ***Figure 1**. A glossary of the many terms in this paper.*

## Modeling agreement systems

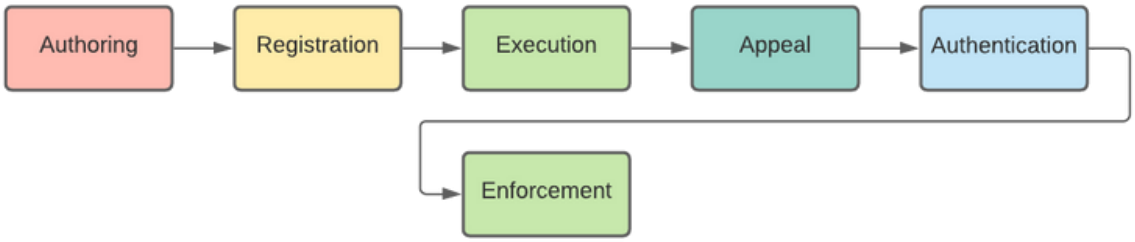
Imagine that you are driving through the streets of Delhi. A worn-down yellow tuk-tuk beeps; you slow, and it merges in front of you. At the next stop light, you wince as a big garbage truck rumbles to a stop just behind your back window. A young chai wallah weaves through the cars with a tray of plastic cups; you wave him over and trade him a few rupees for a cup. The tuk-tuk ahead of you moves on; you accelerate too, swerving past a bicycle vendor on the corner, and turn onto a crowded traffic circle. As you inch forward, a red hatchback edges up at the next turn, signaling right. You slow, graciously; the hatchback moves gingerly up until it works up the courage and enters ahead of you into the wide stream of cars heading downtown.

Every day, we negotiate agreements with each other. We signal; we gesture; we infer; we agree; we act. It is second nature to us. Most of these rich interactions do not produce well-defined artifacts in the form of enforceable contracts. But most *do* take place within systems that support and facilitate them, systems that have

a passing resemblance, if you stare at them closely, to more formal systems like the courts. Our goal in this section is to understand and characterize some of these similarities.

We present *agreement paths*, a formalism for modeling systems of agreements that abstracts over the content and type of an agreement. While agreement paths are general enough to apply to many types of agreements, they are not intended to be a general analytic instrument for scholars of agreements in philosophy, economics, or law. *Agreement paths are intended to highlight the elements of an agreement system that can be digitized.* Our goal is not to present a general model for analyzing or classifying agreement systems but to model the components that are present in digital agreement systems like DocuSign Agreement Cloud, Ethereum, Aragon Court, or Kleros. This way we can more easily (1) recognize these digital agreement systems when they occur on the internet, (2) amend these systems as we amend institutions for agreement in the real world, (3) identify the places where digital processes can enter into offline agreement systems, and (4) build more modular and "swappable" agreement systems.

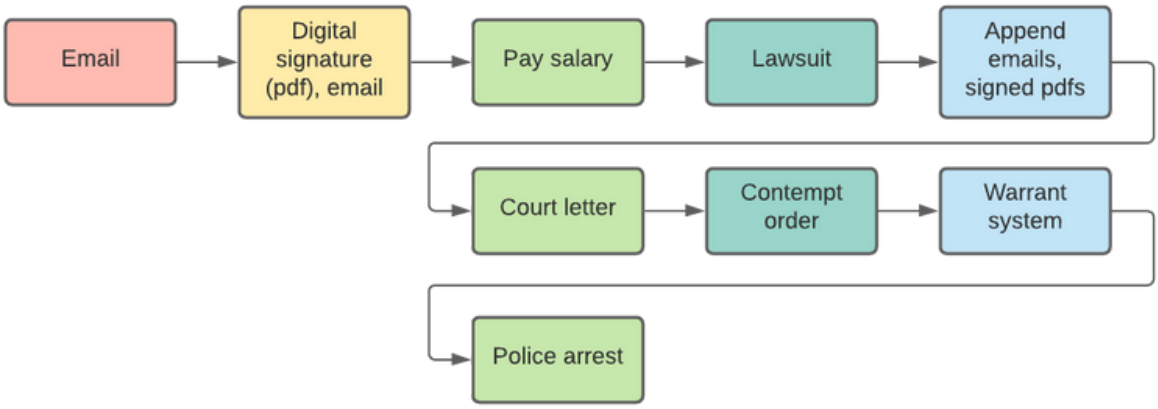
Every agreement path can be broken down into six stages: *authoring*, *registration*, *execution*, *authentication*, *appeal*, and *enforcement*. An authoring process allows parties to negotiate and author agreements with each other. A registration process records the data pertaining to an completed agreement, often including the agreement's identity, content, parties, and status along with secondary authentication data (e.g. the name of a witness). Once registered, the agreement is executed or enacted by the parties to the agreement. During execution, a party may trigger an appeal to exit the execution process for whatever reason (this external appeals process should be distinguished from actions defined within the content of the agreement for resolving disputes).[11] Following an appeal, an authentication process verifies the data, validity, and/or registration status of an agreement. Finally, an enforcement process interprets the content of an agreement, often but not always with accompanying evidence, in order to enforce its execution or compensate for any outstanding obligations. See *Figure 2*.




> ***Figure 2.*** *The six stages of an agreement path.*

For example, Alice and Bob might negotiate the terms of an employment contract over email, trading versions back and forth (authoring). Once they are both satisfied, they convert the final agreement to a PDF and each sign it digitally, trading the signed and counter-signed copies over email (registration). With the contract

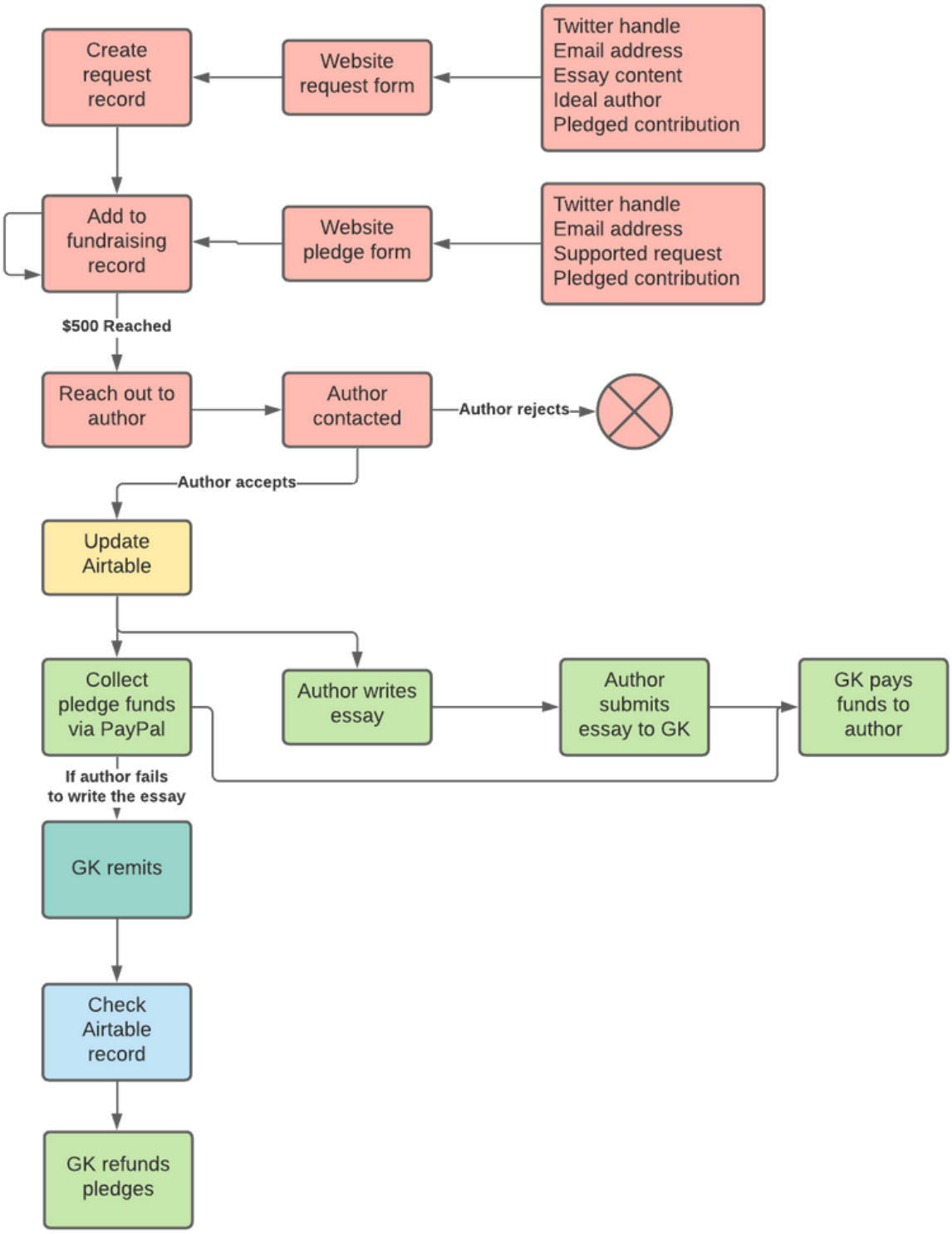
signed, Bob begins working for Alice, and in return Alice pays Bob a salary, as per the contract (execution). At some point, Alice stops paying Bob and stops responding to calls or email; Bob is forced to file a lawsuit to get back his wages (appeal). As part of this lawsuit, he appends copies of the emails and the signed employment contracts (authentication). The lawsuit enters the legal system. A small claims court rules against Alice, who does not show up to contest the lawsuit, and orders Alice to pay Bob (enforcement) by sending a letter to Alice's last-known address. Alice does not respond to the letter, at which point the trial court issues a contempt order for Alice to be arrested (appeal).




> ***Figure 3.*** *A simple employment contract negotiated over email and enforced via a trial court.*

Not all agreement paths are as straightforward as the one above.

For example, Ghost Knowledge[12] is a website that allows people to crowdfund "bounties" for specific authors or researchers to write essays. Once pledges go past a certain threshold, Ghost Knowledge messages the authors on social media, who then either accept or reject the bounty; if they accept, the money is put into an escrow account to be released once the essay is published.

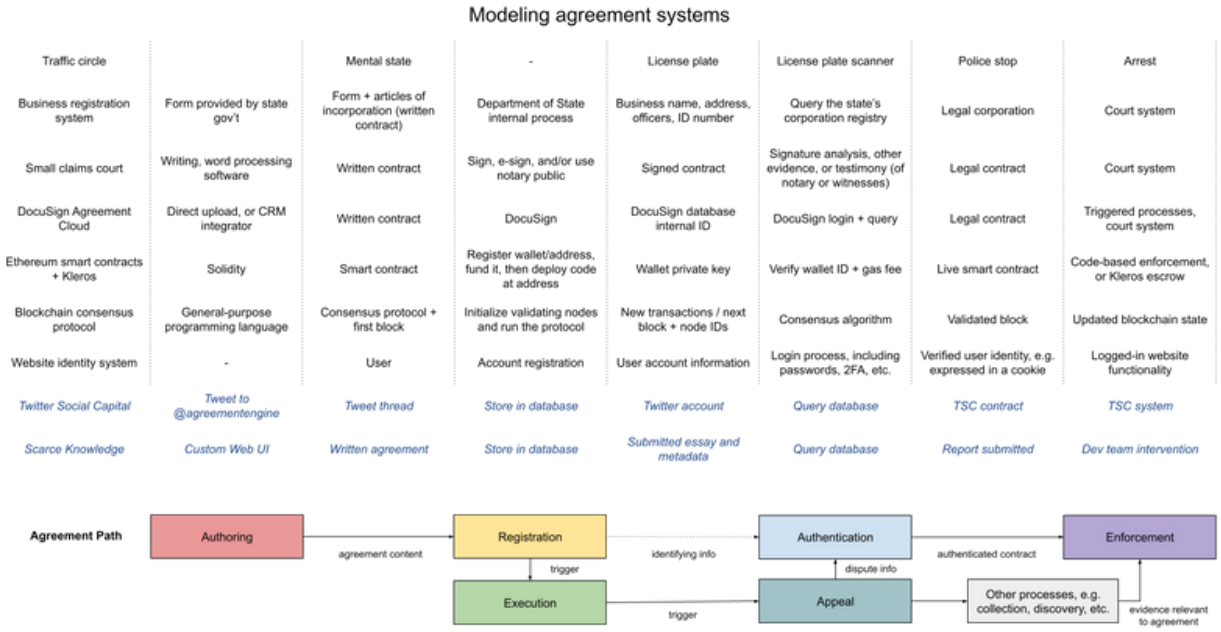



> ***Figure 4.*** *The agreement path of Ghost Knowledge, a website for crowdfunding written essays.*

Ghost Knowledge was built and released using no-code tools including Airtable, PayPal, and the Twitter API. A substantial portion of the author interactions involved manual messaging by the founders.[13] We will come back to Ghost Knowledge in the next section, when we recreate a version of it using the Agreement Engine tool.

| Scope | Definition | Examples |
|---|---|---|
| Authoring | An authoring process allows end-users to negotiate and author agreements with each other. | Ghost knowledge negotiation (No-code form + Twitter DMs) |
| Registration | The process of recording or representing a completed agreement and related data, usually by a qualified authority | Signing a physical document, clicking accept on a digital form |
| Execution | A special layer of the enforcement process related to the content of an agreement. Includes things such as enactment, delivery, and any number of triggers specified in the content of the agreement, e.g. revisions, termination clauses, and so forth. | Payment, sending of goods, execution of contract calls |
| Appeal | A process that triggers an authentication and enforcement process. Appeals can be pre-specified within an agreement, but they always call an external process that is not specified in the agreement itself. | Communication between parties to revise a contract.<br><br>*Non-example: an adjudication clause specified in the contract itself.* |
| Authentication | The process of checking the data, status, and/or registration of an agreement, usually by an enforcement mechanism | Admission of evidence in a courtroom, reporting a video on YouTube |
| Enforcement | An enforcement mechanism interprets the content of an authenticated agreement, often (but not always) with accompanying evidence, in order to enforce or compensate for any outstanding obligations. | Kleros jury, U.S. justice system, moderator intervention on a social media platform |

> ***Figure 5.*** *Definitions of the stages of an agreement path.*




> ***Figure 6**. Additional examples of the key processes in an agreement system:* authoring, registration, execution, appeal, authentication, and enforcement.

## Constructing and composing agreement paths

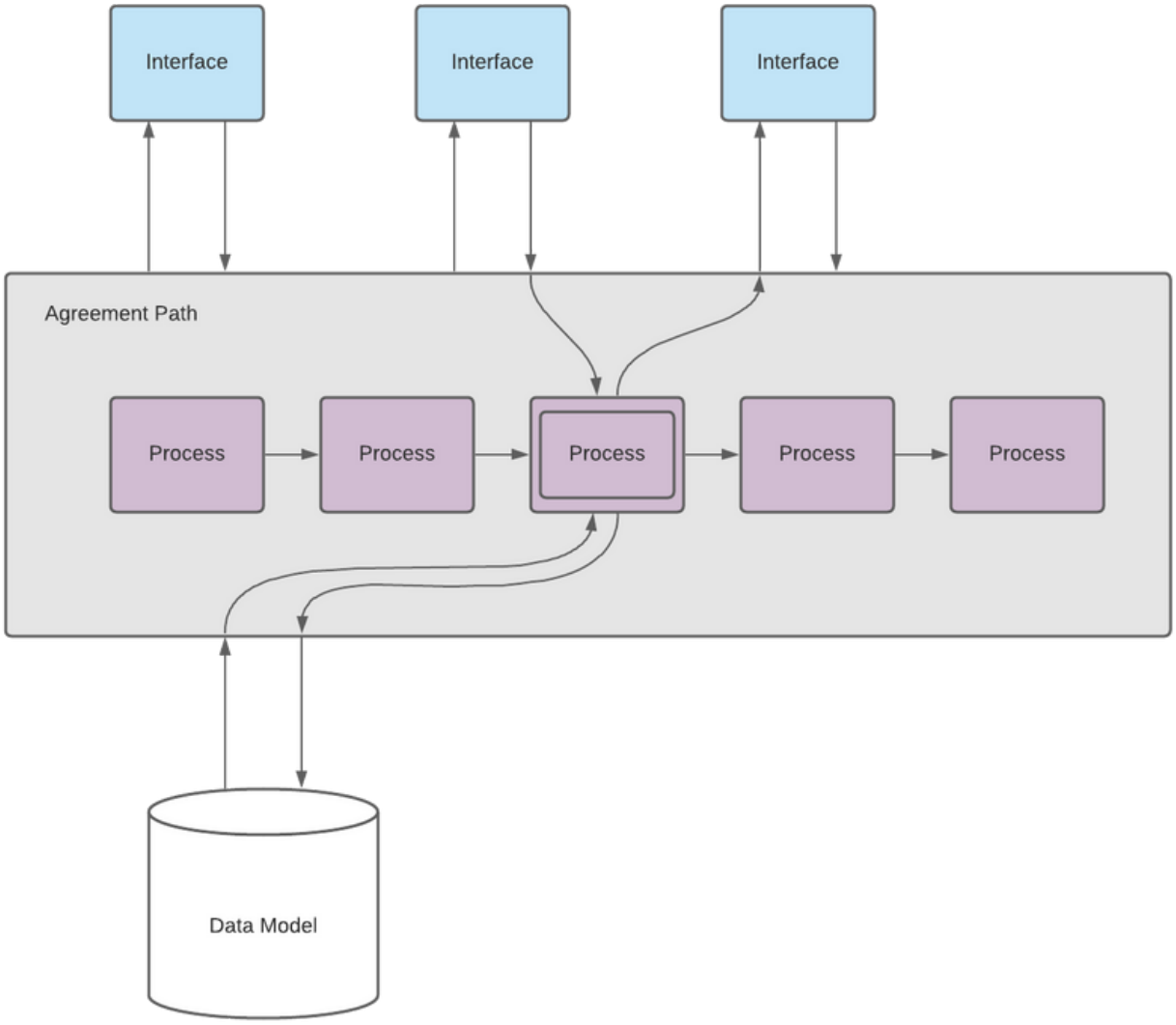
Agreement paths, on their own, are a conceptual model intended to support the composition and substitution of different processes within agreement systems. In this section, we present Agreement Engine, a software tool for building net-native agreement systems. Agreement Engine implements the agreement path model.

To be clear, Agreement Engine itself is neither a system for authoring individual agreements nor a system for enforcing them. However, it has built-in interfaces with existing authoring and enforcement systems.

### Agreement processes

From a top-down perspective, each agreement system created in Agreement Engine can be considered a state machine composed of many processes. Each process represents a distinct and independent service: authoring, registration, authentication, and so on. Within a given path, not all processes need to be computational, as long as they have a formal representation. When a process is active, all data input to the agreement is routed to it for processing. The process can then decide to wait for additional input, transition to another process, or terminate the agreement. Processes can also take action immediately when they are activated or right before the process ends, instead of asynchronously upon receiving input. Using this method, we can chain discrete processes into agreement paths which exhibit more complex behavior.

In our archetypical agreement system model, this starts with an *Authoring* process and ends with an *Enforcement* process.

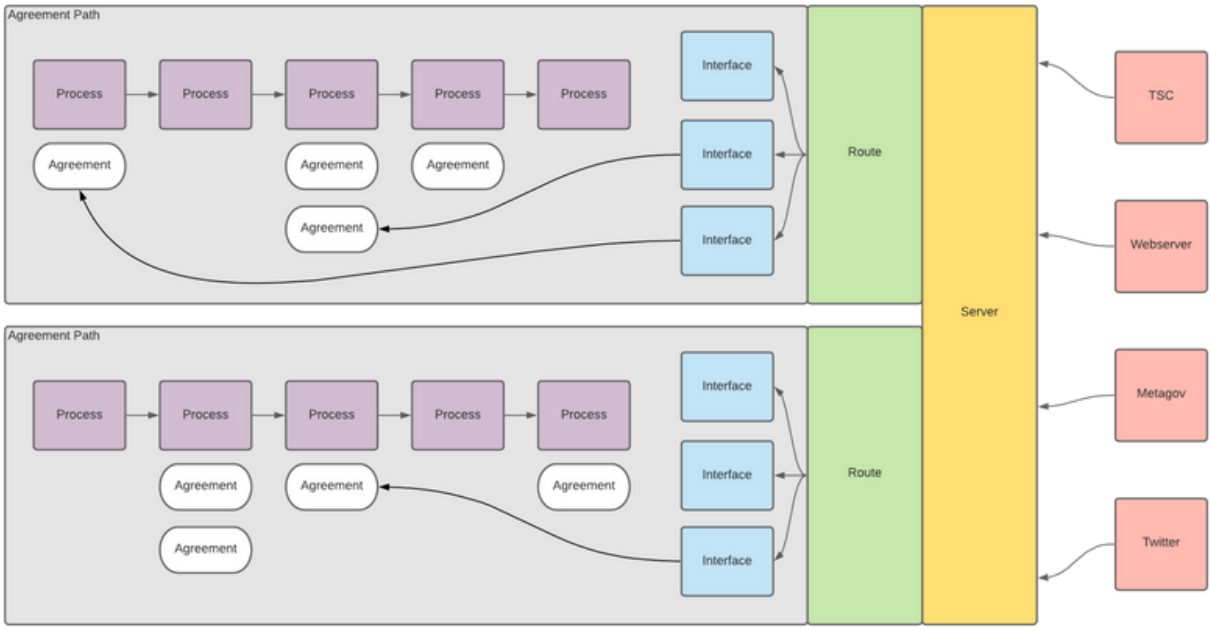



> ***Figure 7.*** *Diagram of a single agreement instance and connected systems*

## Agreement interfaces and instances

Within Agreement Engine, agreement paths are high-level data structures that model a single agreement system along with an arbitrary number of agreements authored within that system. Paths can manage processes by keeping track of which process is currently active, handling state transitions, and providing access to each agreement's private data model. Agreement paths also filter and redirect agreement input and output via a set of agreement interfaces. Interfaces filter incoming data from various sources so that it can be preprocessed and routed to the correct agreement instance (or used to create a new agreement instance). As seen in Figure 7, a single instance of an agreement path, or simply referred to as an agreement instance, acts as a passthrough for the currently-active process. It allows the agreement to be viewed as a single object from the outside, while facilitating more complex decision-making from the inside. Each agreement instance has its own state and data

model in order to track the path it is associated with, the stage within that path, and any additional data relevant to its execution. Paths can manage any number of agreements, as seen in Figure 8.




> ***Figure 8.*** *Diagram of a server with multiple paths and active agreements*

## Agreement servers

Agreement servers allow for multiple paths (and thus multiple agreement instances) to be run at the same time. The server receives requests and routes them to the corresponding path where the interfaces decide which agreement instance should receive the data. This system allows for arbitrary data to be handled, whether it's from a local process, a webhook, or another web server. The server provides the infrastructure for agreements to connect to the internet and interface with other platforms and software. For example, suppose we have a server running on a web server with the domain name "*myagreements.com*." We could configure a new agreement path called "*Employment*" for managing employment contracts. Now our agreements can receive data sent to "*myagreements.com/Employment*" via REST API calls from other applications, or webhooks from platforms such as Twitter or Slack.

## Example: Scarce Knowledge

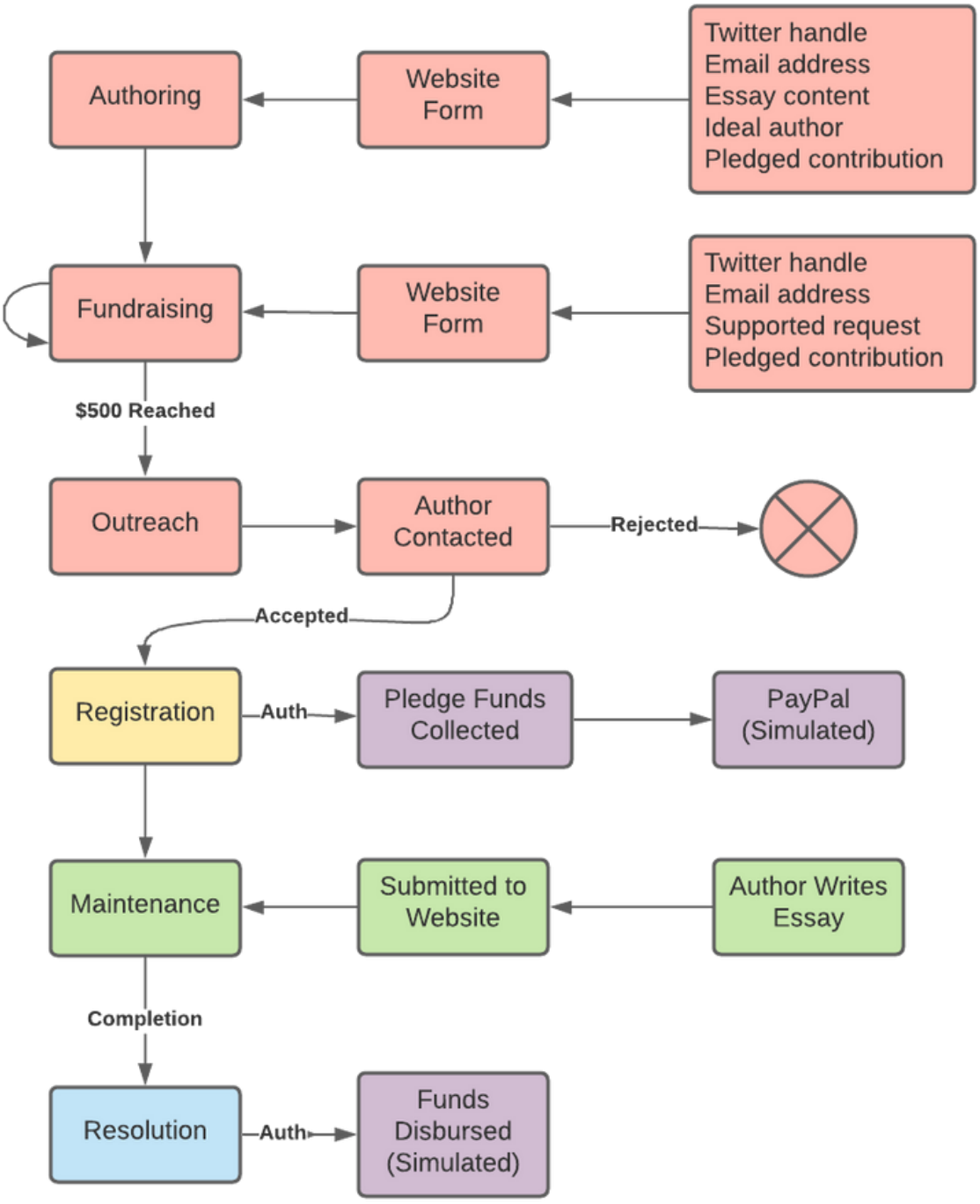
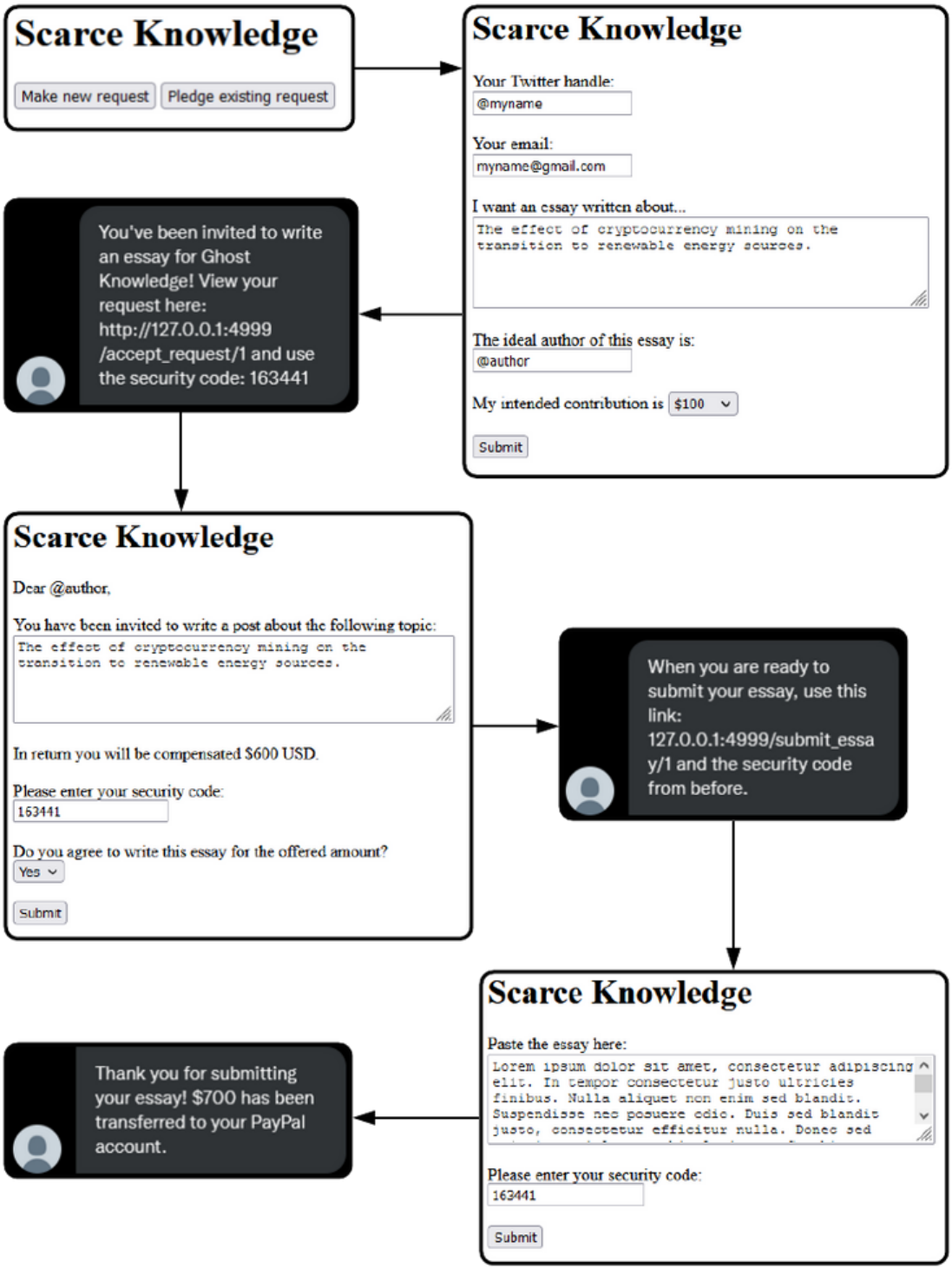


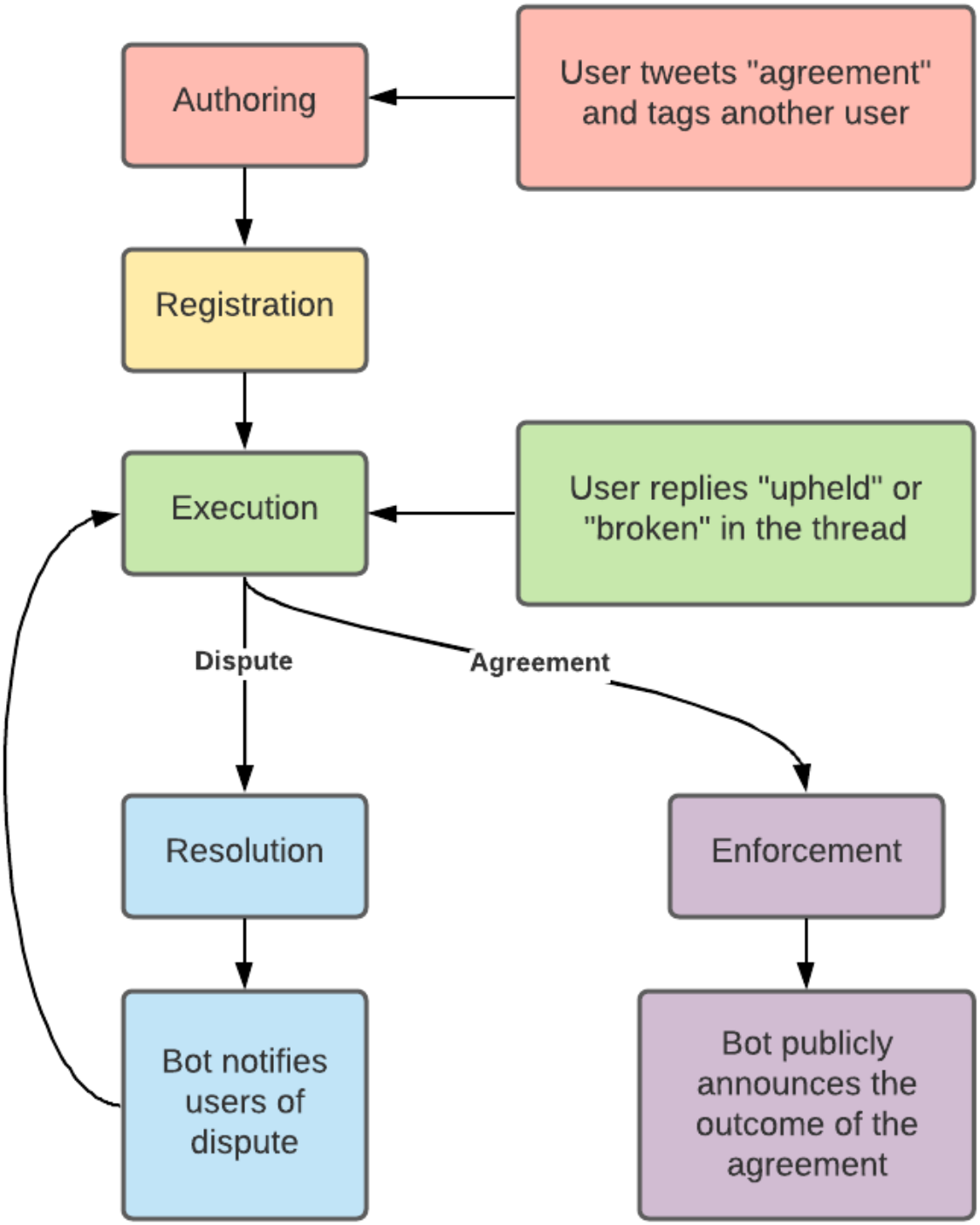

***Figure 9.*** *The agreement path for Scarce Knowledge*

Let’s look at an example agreement system built using Agreement Engine. This demo, which we call Scarce Knowledge, recreates a version of the Ghost Knowledge platform that we described in the previous section. Figure 9 shows the agreement path of this version. Requests to Scarce Knowledge are initiated by a user-facing website with a simple form that forwards data to Agreement Engine. Once pledged contributions surpass $500, a process transition is triggered. This is done via a Twitter interface that sends a direct message to the requested author. The next few processes return to the website to handle final essay submission. Finally, an email automation service is called to forward the essay to those who supported it. By using Agreement Engine, we were able to manage a complex decision-making process with several approval steps that span a user facing frontend, Twitter, and a mail automation service. In a production implementation this would also include PayPal integration. Figure 9 below shows part of the user interface flow for this agreement path implementation using a simple website and Twitter direct messages.



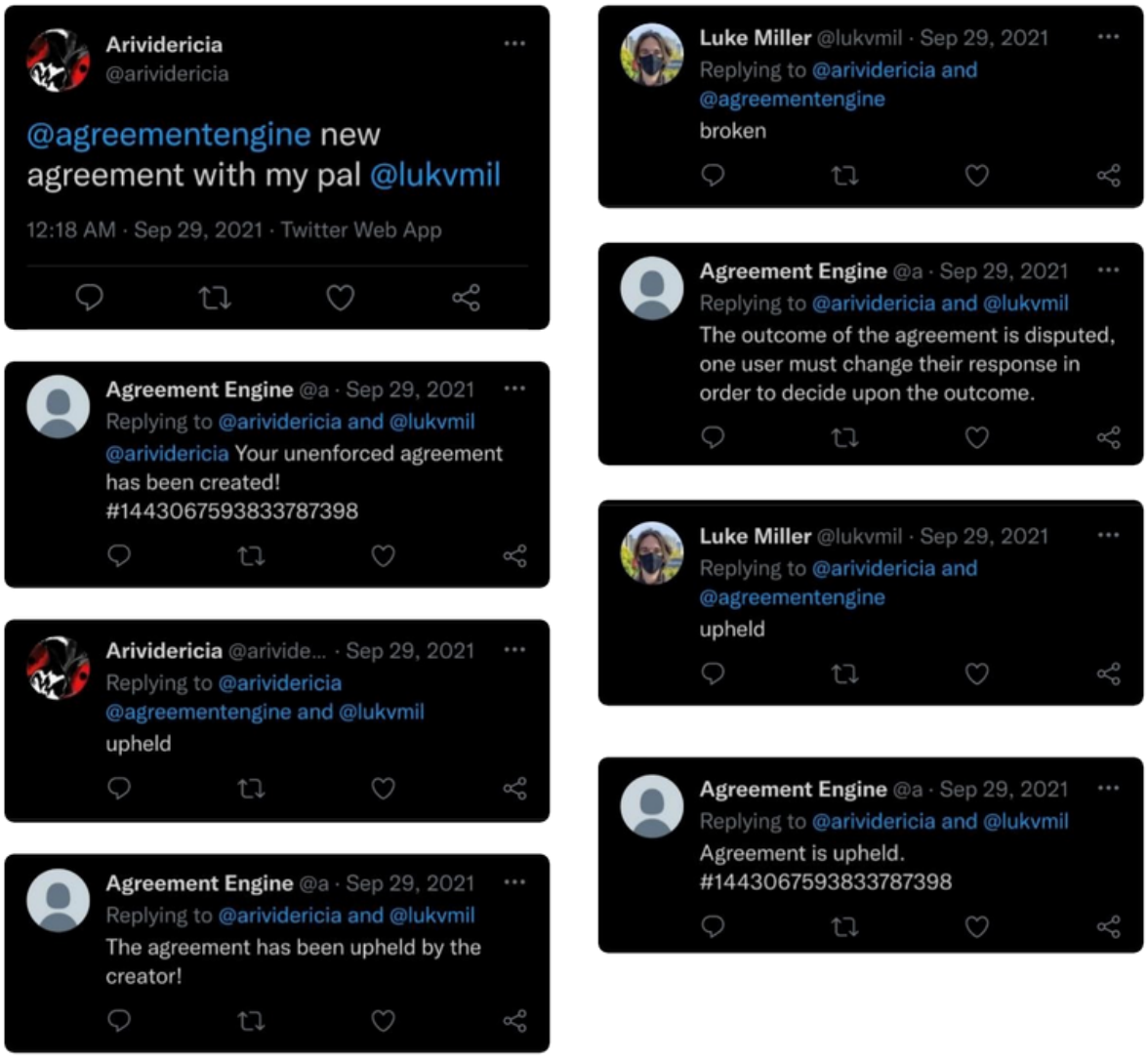

> ***Figure 9.*** *User interface diagram for "Scarce Knowledge"*

## Example: Twitter Social Capital




> ***Figure 10***. *The agreement path for Twitter Social Capital (TSC)*

Let's look at another example agreement system implemented in Agreement Engine. Twitter Social Capital (TSC) is a Twitter-based agreement system that allows users to register and enforce agreements with each other by tweeting at the @agreementengine bot. New agreements are authored by tagging another user and using the keyword "agreement" Users can then reply to the tweet with "upheld" or "broken" to indicate the status of the agreement. If users disagree on the outcome they are given the chance to change their responses and come to a consensus. Finally, the bot publicly records the result of the agreement. As seen in Figure 10, by using Agreement Engine we were able to create a branching and looping control flow to resolve disputes and only terminate when an agreement is reached. This example uses an interface to the Metagov Gateway to send and receive tweets. The technical details of interfaces are discussed later, but this implementation demonstrates the flexibility of Agreement Engine in integrating with other platforms.



> ***Figure 11.*** *Example tweets from the TSC Twitter bot that we built using the Agreement Engine.*

| Feature | Notary public | DocuSign Agreement Cloud | Kleros or Aragon Agreements |
|---|---|---|---|
| Supports authoring | No | Yes, through Gen and other integrators | No |
| For legal consumption | Yes | Yes | No |
| For consumption by online community | No | No | Yes |
| Many enforcement mechanisms | No, just the law | No, just the law | No, just escrow |
| Many authoring platforms | No | Some, e.g. Salesforce | No, just Ethereum |
| Includes decision-making process for enforcement | No | No | Yes, digital jury |

> ***Figure 12****. Comparison of different registration services.*

## Using the Agreement Engine library

When creating an agreement system using the Agreement Engine Python library, there are three main components that should be considered: the server, paths, and interfaces. Let's construct a simple agreement to explore how this works.

```
class MyInterface(Interface):
    def filter(self):
        return True

    def match(self):
        return self.new_agreement()
```

> ***Figure 13.*** *Example interface code*

First we need to define an interface. This is how a path decides whether incoming data should be routed to an existing agreement or a new one should be created. In the example code above, we create an interface that handles all input, and always creates a new agreement. Real applications will decide when different interfaces should govern input data, and how to match existing agreements. For example, the Scarce Knowledge example uses two interfaces to communicate with the user facing web server and the Twitter API.

```
class MyPath(AgreementPath):
    class Authoring(AgreementProcess):
        def on_receive(self, data):
            self.model.set('data', 'received', data)
            self.path.terminate()

    interfaces = [
        MyInterface
    ]

    init = Authoring
```

> ***Figure 14.*** *Example agreement path code*

Next we define our agreement path, this is the primary structure that represents our agreement system. It will contain all of our processes and define the path that an agreement takes through them. In this example we have a single authoring process that records the data we received to the agreement's data model (a simple JSON format with different fields) and then terminates. We also set up the interfaces that the path is using, and the initial process that a new agreement will start in. Paths can implement more complex logical flow such as branches or loops. Our example paths use four processes that can terminate early upon detecting invalid data inputs.

```
server = Server('db.json')
server.add_path(MyPath)
server.run()
```

> ***Figure 15.*** *Example server code*

Finally we create the server that actually receives and sends requests. All we need to do is define a JSON file to store our agreement data and add our paths. Servers can support any number of paths, as long as they have unique names. Each path will receive a unique API URL for receiving requests. The URL for a path on a local server will look like http://*127.0.0.1/MyPath*.

## The challenge of composition and re-use

One of the first lessons we learned was that enforcement mechanisms, or any agreement process, are not as hot-swappable as we would like to think that they are. The Agreement Engine Python library supports reusing processes by "transplanting" them from existing agreement paths into new ones, allowing us to build up a library of quickly usable processes, but there are some limitations. When modeling these systems conceptually, we can abstract away from the exact data being passed around by different functions, but in order to implement the system in software, we need to explicitly state what data is being stored in an agreement, accessed in different processes, or passed to external functions. But this data resists standardization.

To see the problem, imagine a simple example where we want to swap the authoring processes of Scarce Knowledge and TSC. Scarce Knowledge receives new agreements via form submission on a website, while TSC receives new agreements by tweeting at the @agreementengine account. These two methods are both simple and have the same output types (agreements), but problems emerge if we attempt to swap them. The TSC agreement system works by reading tweets that reply to a root tweet, using it to identify unique agreement interactions, and all interactions remain on Twitter. On the other hand, Scarce Knowledge ties together a few different platforms: a simple website, Twitter direct messages, and an email plugin, and the actual content of the agreement has to be synthesized from these platforms. In other words, it is hard to design processes to be completely disentangled from the ones that come before or after them.

Here we come to one of the fundamental tradeoffs: immediate flexibility vs long-term interoperability. Currently, Agreement Engine provides useful software abstractions including the interface and path model which make it easy to set up simple agreement systems. Users don't have to worry about the details of how the web server functions, or how to get data to specific agreements. Instead they can focus on the actual logic of the agreement, and what decisions should be made based on the incoming data. In future versions of Agreement Engine, we want to introduce similar software abstractions to path composition—i.e. other words, we want to standardize the interfaces and possible interactions between agreement stages. From a systems perspective, we believe that the benefit of interoperability between different agreement paths is more important than preserving the short-term flexibility of being able to code whatever logic one likes.

## Discussion

The first version of Agreement Engine was a net-native service for *registering* net-native agreements, especially on Twitter, and in a way it still serves that purpose.[14] It only evolved into a tool for implementing agreement *systems* as we tried to navigate the difficulties of creating, testing, and swapping net-native enforcement mechanisms for these agreements.

We discussed the swapping problem above. But engaging with agreement systems has also led us to a range of more social and less technical questions. Would different systems for agreement foster new solutions to persistent coordination and collective action problems? How do we reason about the tradeoffs between these systems? And by better understanding the properties and affordances of such systems, could we generate new kinds of agreements with novel properties and affordances? And on the practical side: who will be the first users of something like Agreement Engine?[15]

Our approach to engaging with these questions is empirical—through more experiments and better experiments. After all, Agreement Engine is ultimately a tool for generating and experimenting with new agreement systems. Here are some other agreement systems that we would (eventually) like to build with it:

1. A Twitter agreement, enforced by staking one's reputation on Reddit
2. A Markdown file with a code of conduct hosted on GitHub, enforced by a GitHub Group's owner

3. A text contract published via Google Doc, enforced by a jury of Slack users
4. A smart contract *not* on Ethereum, enforced by placing tokens in escrow on Kleros
5. A grant agreement from one DAO to another DAO, enforced by a third DAO

---

**Joshua Z. Tan** is affiliated with Oxford University and Metagov

**Luke V. Miller** is affiliated with Metagov

---

## Acknowledgements

We would like to thank Lawrence Lessig, Nathan Schneider, and Miriam Ashton for helpful comments in the development of this project.

## Footnotes

1. Legal Information Institute, *Agreement*, LII / Legal Information Institute (2021), https://www.law.cornell.edu/wex/agreement (last visited Dec 28, 2021). ↩
2. eBay Inc., *eBay unpaid item policy*, eBay (2021), https://www.ebay.com/help/policies/payment-policies/unpaid-item-policy?id=4271 (last visited Dec 28, 2021). ↩
3. Mathias Reimann & Reinhard Zimmerman, The Oxford Handbook of Comparative Law (2006), https://www.oxfordhandbooks.com/view/10.1093/oxfordhb/9780199296064.001.0001/oxfordhb-9780199296064 (last visited Dec 29, 2021). ↩
4. Robert Merkin & Séverine Saintier, Poole's Textbook on Contract Law (15th ed. 2021). ↩
5. Daniel Markovits & Emad Atiq, *Philosophy of Contract Law*, *in* The Stanford Encyclopedia of Philosophy (Edward N. Zalta ed., Winter 2021 ed. 2021), https://plato.stanford.edu/archives/win2021/entries/contract-law/ (last visited Dec 27, 2021). ↩
6. Patrick Bolton & Mathias Dewatripont, Contract Theory (2004). ↩
7. Elinor Ostrom, Governing the Commons: The Evolution of Institutions for Collective Action (1st edition ed. 1990). ↩
8. Nick Szabo, *Smart contracts: building blocks for digital markets*, 18 EXTROPY J. Transhumanist Thought (1996). ↩
9. Vitalik Buterin, *A next-generation smart contract and decentralized application platform*, https://ethereum.org/en/whitepaper/ (2013), https://ethereum.org (last visited Dec 28, 2021). ↩

10. Jae Kwon & Ethan Buchman, *Cosmos Whitepaper*, https://cosmos.network (last visited Dec 29, 2021). ↩

11. An appeal (or the right to appeal) is a part of the meaning of the agreement that is independent of the "content" of the agreement. It is enabled by the agreement system. ↩

12. Sari Azout, *Ghost Knowledge*, Ghost Knowledge , https://www.ghostknowledge.com/ (last visited Jan 13, 2022). ↩

13. Sari Azout, *Interview with Sari Azout* (2021). ↩

14. There is a subtlety here. Within any agreement path there will be a registration process that tracks whether something is an agreement. This registration may store the tracking data anywhere, including outside of the Agreement Engine. However, for practical purposes an agreement path also needs to represent and track the state of an agreement as it passes between stages. So, in effect, Agreement Engine functions as a registration service. ↩

15. Our hypothesis is that our first users will be communities who want to build customized contract frameworks such as Ghost Knowledge that help produce highly-specific contracts for a particular community. ↩