Regerator: a Registry Generator for Blockchain

An Binh Tran[1], Xiwei Xu[12] Ingo Weber[12], Mark Staples[12], and Paul Rimba[1]

[1] Data61, CSIRO, Sydney, Australia
[2] School of Computer Science and Engineering, UNSW, Sydney, Australia
{firstname.lastname}@data61.csiro.au

Abstract. A registry is a list of information recorded by a trusted authority. Registries have security requirements for data integrity and availability, and for the ability to connect with other registries. Building registries on a blockchain leverages key properties of blockchains, including data integrity, immutability, and availability. By using a blockchain as uniform infrastructure, different registries can also more easily interact with each other. In this paper, we present a browser-based tool for the model-driven generation and deployment of registries as smart contracts on blockchain. The tool also generates web-based RESTful APIs and user interfaces to interact with the generated registries. We evaluate the feasibility and transaction costs for this approach using metadata from data.gov.au, stored on a decentralised derivative of CKAN (a web-based open-source data registration system) built on the Ethereum blockchain[3].

Keywords: Blockchain, registry, code generator, smart contract

1 Introduction

A registry is a list of information recorded and managed by a trusted authority. For example, a government might maintain a registry to store information about businesses, including their business number and name. However, a centralised service maintaining a registry is a single point of failure for the whole system. One approach to address this is to use a blockchain [4] for the registry, which is an emerging technology for secure, decentralized and transactional data sharing across a large network of untrusted participants without relying on a central trusted authority to record and validate transactions.

The blockchain data structure is a time-stamped list of blocks. The blocks are chained together cryptographically: each block is digitally signed and "chained" to the previous block by including that block's hash value. New blocks are only appended to the end of the chain, thus the blockchain provides an immutable data storage: existing transactions cannot be updated or deleted. The immutable chain of historical transaction provides *non-repudiation* of the stored data. Cryptography and digital signatures are used to prove identity and authenticity and to enforce read and write access control to the blockchain. A blockchain network

[3] A screencast video of this demo can be found at `https://youtu.be/vDj2yoX8Ois`.

relies on *miners* to aggregate transactions into blocks and append them to the blockchain. Every transaction within the new block is verified by participants of the network to ensure *integrity*. The whole network reaches a consensus on whether a new block is included into the blockchain.

Building registries on blockchain can provide increased confidence in data integrity, availability, transparency and immutability, and there is strong interest from industry and government around this idea. In particular, data integrity and availability are two of the key requirements of registries [2]. Additionally, if we use a blockchain as a unified infrastructure, multiple registries can more easily interact with each other. There are registries being built on blockchain in ad-hoc ways, for example, Namecoin[4], which is a domain name registry that shares the same network with Bitcoin[5], and Abscribe[6], which is an artwork registry that allows artists to register and manage the ownership of their digital artwork. However, building a registry on blockchain is non-trivial due to the steep learning curve of the technology [1]. Regis[7] is a contract generator on Ethereum[8] blockchain, but only provides very basic operations.

In this paper, we present *Regerator*, which is a tool that follows a model-driven approach to provide templates for the developers to customize registries and automatically generate and deploy registries on blockchain. We use a web form and a model that is less bound to the underlying blockchain technology. Regerator includes 1) a smart contract generator that can generate and deploy smart contracts representing registries on Ethereum blockchain, and 2) a generator for web-based RESTful APIs and user interfaces to interact with the generated registries. The envisioned users of our tool are developers with limited knowledge of blockchains or smart contracts. The feasibility of the tool is illustrated through a study of an open data registry, using meta-data from `data.gov.au`, and a registry model derived from the CKAN platform.

2 Registries on Blockchain

Registries are authoritative databases for specific entities and are used to manage many aspects of daily life, such as land titles, business names, books, marriages, births and deaths, music, films and domain names. Many public registries are hosted and maintained by government agencies whose authority guarantees authenticity for the registered entities. Every change to a registry is recorded with a digital fingerprint, which can be verified independently. A registry should store a history of all changes and be open to independent scrutiny. A registry may reference other registries to reduce duplication and errors. Registries should be highly available, because other registries and services depend on them [2]. Open registries are publicly available, which means that the registry may be accessed,

[4] `https://namecoin.org/`
[5] `https://bitcoin.org/`
[6] `https://www.ascribe.io/`
[7] `https://regis.nu/`
[8] `https://www.ethereum.org/`

copied, or derived freely by the public. For instance, a business name registry, such as the Australian Business Register[9], is a public registry whose entities can be requested by anyone at any given time. Building registries on blockchain can leverage key properties provided by blockchain and utilise the infrastructure of blockchain to achieve interoperability.

Integrity concerns the accuracy and consistency of data over its entire life-cycle. Data integrity is a key requirement of a registry, which means that the items can be only registered and changed by the authorized users. Many blockchain techniques are censorship-resistant, which helps to ensure the ongoing integrity of the full log behind the registry.

Availability is also a key requirement for registries, especially national public registries, which form the basis for many other services that utilize the data from the registries. A blockchain system maintains consensus on data that is replicated across the network with many processing nodes, so that there is no single point of failure since the infrastructure is fully decentralized.

Interoperability is achieved since blockchain provides a universal infrastructure for registries to easily refer to and interoperate with each other.

Efficient reading is achieved because every node within a blockchain network has a local copy of all historical data. This allows large-scale users of the registry to access local copies of the registry directly, which may reduce latency and cost. However, light users might find the cost of operating a full node relatively high, e.g., when compared to API calls.

Programmability is provided by *smart contracts* which allow the implementation of more sophisticated, flexible, and finer-grained access control models to register and manipulate the items in the registry. Smart contracts are programs that can be deployed and running across the blockchain network [3]. Smart contracts can express triggers, conditions, and even an entire business process. And the computational results are verified by the participants of the network and recorded on blockchain. For example, transferring ownership of items can be easily implemented in smart contracts. Ethereum is the most popular second-generation blockchain.

Immutability is another key property of blockchain. On one hand, immutability enables an audit trail of all the historical operations on the registry, so there is complete traceability of records. On the other hand, some registries need to be able to remove records from the registry as if those records were never created, e.g., since their creation violated legislation.

3 Regerator

Regerator is a model-driven framework for the generation of registries on a blockchain, and for the generation of interfaces to those registries. Currently it generates registries for Ethereum and Solidity (a smart contract language). However, as a model-driven framework it could potentially support additional

[9] `https://abr.gov.au/`

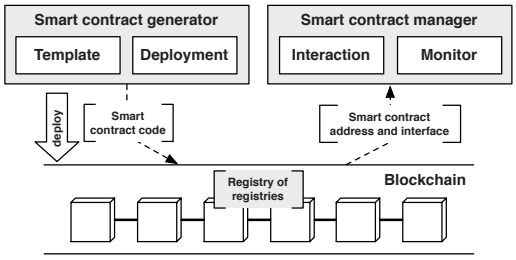

Fig. 1. Overview of Registry Generator on Blockchain

back-end blockchain platforms in future, provided that those platforms have sufficiently-expressive smart contract languages. Regerator has three core components: a *smart contract generator*, a *registry of registries*, and interfaces for *smart contract management*, as shown in Fig. 1.

3.1 Smart Contract Generator

The smart contract generator allows the users of Regerator to generate smart contract registries from registry models, and to deploy the generated smart contracts onto the blockchain. The smart contract model has four parts, including *basic information*, *registry type*, *basic operations* and *advanced operations*.

Basic information includes the registry name, description, and user-defined data fields and their types.

Registry type can be 'single' or 'distributed'. The 'single' registry type holds all records as values in the data store for a singleton smart contract for the registry. The 'distributed' type manages each record as a separate smart contract. A main registry smart contract creates these contracts and stores pointers to them. The 'single' option is suitable for simple registries, while the 'distributed' option is suitable for registries with complex operations, such as finer-grained permission management at individual record level.

Basic operations are the operations that can be performed on an individual record, including Create/Read/Update/Delete and existence checking. Users could configure whether or not a record is updatable.

Advanced operations include *access control*, *foreign key*, *version control*, *provenance*, and *trading*. We explain them in more detail below.

Access control is required to restrict users to certain operations. In the case of a public registry, only authorised government agencies are allowed to insert or update records, even though the registry is readable by the public. To enable permission management, a whitelist or a blacklist of addresses can be provided for the invocation of operations. We allow for definition of access control mechanisms at the registry layer or the record layer. We provide two types of access control management. The basic type is to check the permission directly before executing an operation. The second type is to use a separate indirectly-invoked

permission smart contract as a gateway to manage the whitelist or the blacklist; the operations of the registry then only check against the address of the permission contract. Deciding between these two alternatives depends on several factors, such as coupling, modifiability and the size of the smart contract, which impacts the cost of deployment. **Foreign key** is a concept borrowed from relational database, which allows users to include the identity of a record from one registry as an attribute of a record to another registry as a way to define the relationship between two registries. **Version control** allows users to explicitly add a version number to a certain update on a registry and enables more efficient query. **Provenance** in the context of registry refers to a log of all the operations that have been executed on a given registered entity. Such information is necessary for auditing data integrity. Blockchain-based registries naturally support provenance, as all data on the blockchain is immutable and valid. **Trading/transferring ownership** is required by registries that allow for trading registered items, such as domain names registered in Domain Name System (DNS). This function is implemented as an escrow, which holds the money from the buyer first, and then transfers the money to the current owner of the item after changing the ownership. **Multi-Signature** requires multiple parties to jointly sign a transaction to invoke a smart contract operation. For instance, a publication registration like *arXiv.org*[10] might require the permissions from all the authors of an article to update or delete the record. This function is planned for future work.

After registries have been defined, the smart contract generator provides a view to show the registries and the relationships among them in a model. Users can then deploy the registries on blockchain.

3.2 Registry of Registries on Blockchain

The registry of registries stores all the registries generated using Regerator on-chain. This facilitates version control of the generated registries. If a registered registry is updated to a new version, the developer needs to add the address of the new smart contract to the registry of registries. Users could query the registry of registries to check the current status of a registry or retrieve a historical version.

3.3 Smart Contract Manager

The smart contract manager provides web-based RESTful APIs and user interfaces to allow users to manage and interact with the generated registries. For each of the functions defined in a registry, there is a dry-run mechanism that validates and tests the transaction by invoking the function on the local blockchain node behind the interface. If the output of the dry-run matches the user's expectation, the transaction is submitted into the blockchain network. This dry-run mechanism allows users to check the effect of their transactions before making permanent changes and incurring actual cost for submitting the transactions

[10] https://arxiv.org/

to the blockchain network. A smart contract monitor provides functionality to monitor contract events. In Ethereum, smart contracts can emit events and write logs to the blockchain when a transaction is processed. The users can watch for new events, which show up on the page when there are events being recorded on blockchain during the contract execution.

4 Exemplar Case Study: Open Data Registry

To demonstrate the feasibility of our approach for model-driven generation of blockchain-based registries, we used Regerator to build a metadata registry inspired by the Comprehensive Knowledge Archive Network (CKAN[11]). We populated this example registry with metadata taken from `data.gov.au`. We discuss some design considerations from the implementation, and discuss transaction cost below.

4.1 CKAN

CKAN is a web-based open-source data registration system, which provides functionalities to streamline publishing, sharing, finding, and using of data. CKAN has been used by public institutions and governments to open their data to the general public, e.g. `data.gov.au` and `data.gov.uk`.

The central entity type in CKAN is a *package*. A *package* defines a variety of metadata of datasets, such as name, description, license, and tags. CKAN also supports an unlimited amount of customized metadata in the form of key/value pairs. The relationships between packages can be defined, such as *depends on*, *child of*, and *derived from*. Another entity type in CKAN is *resource*, which represents the raw data in the dataset, such as files or APIs. A *package* can be associated with multiple *resources*.

4.2 Implementation

We modelled elements of CKAN's metadata schema using Regerator, and generated a blockchain-based registry system for the metadata of datasets. One architectural decision to be made is either to manage one entity as part of the attributes of another entity or to model both entities as separate registries. For the first choice, the nested entity will not have a unique, identifiable ID. As for the second choice, *Foreign Key* references between them need to be defined in order to encode the relationship and both the entities can be uniquely identified. For the entity to be modelled as registry, another architectural decision to be made is either to model the entity as a 'single' registry or a 'distributed' registry. The factors to consider include the complexity of the data structure, the nature of the relationship between entities (coupling), and the cost of deploying and executing the registries on blockchain.

[11] `http://ckan.org/`

In the case of CKAN, there are potentially three entities that could be implemented as separate registries, including *package*, *resource*, and *organization*. Although *resources* are associated with a *package*, a *resource* is also a independent entity with its own metadata and can be managed separately. Thus, we have decided to record *resources* in a separate registry. Finally, *organization* is implemented as a separate registry that groups the address of all the users from the same organisation. The organisation registry can be used to define access control, akin to Role-based Access Control.

4.3 Example Data

After implementing the blockchain-based registry, we queried the metadata of all the datasets from `data.gov.au`[12], and added that into our registry, to test the feasibility of our approach. Information about the number of each entity and the collected fields are shown as below.

- Organization(*533*): *name*, *jurisdiction*, *spatial_coverage*, *email*, *telephone*, *website*
- Package(*33810*): *name*, *owner_org*, *license_id*, *contact_point*, *spatial_coverage*, *temporal_coverage*
- Resource(*64147*): *name*, *url*, *package_id*, *format*, *hash*, *size*

During the metadata import, we collected data about the blockchain cost as gas consumed (i.e. transaction execution cost) for deploying a registry and adding a record to the registry. We use this information to calculate the monetary cost of using blockchain as metadata repository according to the cost model [5]. Table 1 reports the cost for the different design options ('single' or 'distributed' registry). The data also shows how different architectural decision can affect the cost of deploying and executing the registry. We assume the gas price is 5×10^{-8} Ether (the default gas price of Homestead as of January 2016[13]) and the price of Ether is US$10 per Ether as of November 2016.

Table 1. Cost of using blockchain

Entity	Registry deployment				Record creation (average)			
	Gas cost		Real cost		Gas cost		Real cost	
	Single	Distr	Single	Distr	Single	Distr	Single	Distr
Organization	1836926	2542604	US$0.92	US$1.27	183266	931179	US$0.09	US$0.47
Package	1836926	2542540	US$0.92	US$1.27	340022	1090174	US$0.17	US$0.55
Resource	1777127	2548455	US$0.89	US$1.27	302041	1065760	US$0.15	US$0.53

4.4 Discussion

Impact of architecture design on cost. On the Ethereum blockchain, the cost of creating a registry contract is comprised of fixed costs and variable costs.

[12] Retrieved at 2017-03-07 15:59:32 AEST (+10)

[13] `https://github.com/ethereum/homestead-guide/blob/master/source/contracts-and-transactions/account-types-gas-and-transactions.rst`

Fixed costs are the base amount for the transaction itself and the cost for allocating an address on the blockchain. Variable costs are affected by the architectural design of the registry contract, for example, the cost of data payload. Similarly, the cost of adding records to a registry is also comprised of a fixed cost for the transaction itself, and some variable costs including for the data payload and to execute the functions defined in the registry contract.

In contrast to existing practice, using a public blockchain means that adding a record costs real money. However, the blockchain ecosystem will retain this data indefinitely as long as the blockchain exists, at no additional cost. The most costly field (with the biggest size) of both *package* and *dataset* in our experiment was "description", which amounted to approx. 85% of the total cost if included on blockchain. If it is not of high importance to store this information on-chain, storing it off-chain could significantly reduce the cost.

Interoperability. In the ecosystem of CKAN, the datasets in different CKAN repositories refer to each other through importing the metadata from the referred repository to the primary repository and transferring it to the correct format due to the customer-defined fields. Regerator allows references to be defined as foreign keys.

5 Conclusion

The model-driven approach is well established, and we show how it can be used for blockchain-based systems. The Regerator system allows users to configure a registry model in a browser-based application and to automatically generate and deploy smart contract code implementing the registry on a blockchain. In addition, Regerator can also create user interfaces and RESTful APIs. Execution cost for a generated registry is affected by architectural options represented within the registry model, and we have explored this through experiments on the Ethereum blockchain. The cost model for blockchains is different from conventional (cloud or in-house) servers, because data is retained indefinitely at no additional cost. In future work, we plan to explore model-driven generation of functions for access control and registry inter-relationships.

References

1. L. Anderson, R. Holz, A. Ponomarev, P. Rimba, and I. Weber. New kids on the block: an analysis of modern blockchains. *CoRR*, abs/1606.06530, 2016. `http://arxiv.org/abs/1606.06530`.
2. P. Downey. The characteristics of a register, 2016. `https://gds.blog.gov.uk/2015/10/13/the-characteristics-of-a-register/`.
3. S. Omohundro. Cryptocurrencies, smart contracts, and artificial intelligence. *AI Matters*, 1(2):19–21, Dec. 2014.
4. M. Swan. *Blockchain: Blueprint for a New Economy.* O'Reilly, US, 2015.
5. G. Wood. Ethereum: A secure decentralized generalised transaction ledger — homestead draft. Technical report, 2016.