DOI: 10.1002/isaf.1424

RESEARCH ARTICLE

WILEY

Toward an ontology-driven blockchain design for supply-chain provenance

Henry M. Kim[1] | Marek Laskowski[2]

[1] Schulich School of Business, York University, Toronto, Ontario, Canada

[2] York University, Toronto, Ontario, Canada

Correspondence
Henry M. Kim, Schulich School of Business, York University, Toronto, Ontario, Canada.
Email: hmkim@yorku.ca

Summary

An interesting research problem in our age of Big Data is that of determining provenance. Granular evaluation of provenance of physical goods (e.g., tracking ingredients of a pharmaceutical or demonstrating authenticity of luxury goods) has often not been possible with today's items that are produced and transported in complex, interorganizational, often internationally spanning supply chains. Recent adoptions of the Internet of Things and blockchain technologies give promise at better supply-chain provenance. We are particularly interested in the blockchain, as many favored use cases of blockchain are for provenance tracking. We are also interested in applying ontologies, as there has been some work done on knowledge provenance, traceability, and food provenance using ontologies. In this paper, we make a case for why ontologies can contribute to blockchain design. To support this case, we analyze a traceability ontology and translate some of its representations to smart contracts that execute a provenance trace and enforce traceability constraints on the Ethereum blockchain platform.

KEYWORDS

blockchain, distributed ledger, enterprise modeling, Ethereum, ontology, provenance, smart contracts, supply-chain provenance, traceability

1 | INTRODUCTION

An interesting practical and theoretical problem in our age of Big Data is that of determining source of information. One community of researchers interested in addressing this problem is the ontological engineering community, who are actively researching the development of ontologies for knowledge provenance (Erickson, Sheehan, Bennett, & McGuinness, 2016; Fox & Huang, 2005).

According to *Merriam-Webster* (2016), provenance is "source or origin; or, the history of ownership of a valued object or work of art or literature." The ontological engineering community's efforts at formally representing and reasoning about the provenance of knowledge on the World Wide Web can be considered tractable because data required to ascertain provenance is in digital form—as data, metadata, and timestamps, for example. Moreover, semantic Web technologies facilitate the semantic and workflow modeling and inference required for Web knowledge provenance. Arguably, provenance evaluation of artifacts that do not have such a ready and openly accessible digital footprint or facilitating technologies has not been as tractable a problem to address. Tracking the ingredients of a pharmaceutical or demonstrating authenticity of a luxury handbag are some examples. Whereas it is true that UPS can accurately track its packages, such granular provenance evaluation has often not been possible with today's items that are produced and transported in complex, interorganizational, often internationally spanning supply chains.

As of late, however, new technologies, namely the Internet of Things (IoT) and blockchain technologies, promise to offer provenance even in complex supply chains (Armstrong, 2016). Internet-aware sensors capture finely granular real-time data about product and environment characteristics as well as location and timestamps throughout the supply chain. So, lack of a digital footprint may no longer be an issue. Furthermore, distributed, shared databases using blockchain technologies promise to offer highly secure and immutable access to supply chain data. Blockchain databases are decentralized, so that provenance can be evaluated even when no one party can claim ownership over all supply-chain data. Inasmuch as metadata and semantic Web technologies enabled ontologies to be applied for knowledge provenance, it is a key premise of our research that IoT and the blockchain, in particular, now can enable ontologies to be used for much improved supply-chain provenance. Armed with this premise, this paper details our efforts toward developing an ontology-based blockchain for supply-chain provenance.

The paper is organized as follows. In Section 2, we expound the blockchain, which constitutes the enabling technology for our work. Excerpts of the TOVE Traceability Ontology, which serves as the ontology source for our blockchain, are presented in Section 3. Following this, Section 4 presents a proof-of-concept implementation of a provenance evaluating blockchain executed on the Ethereum application development platform and encoded in the Solidity language. Finally, in Section 5 we present concluding remarks and commentary for future work.

2 | THE BLOCKCHAIN

According to a technical definition, a blockchain is "a distributed database that maintains a continuously-growing list of data records secured from tampering and revision. It consists of blocks, holding batches of individual transactions. Each block contains a timestamp and a link to a previous block" (Morris, 2016; Nakamoto, 2008; Popper, 2016). This cryptographic technology "offers a way for people who do not know or trust each other to create a record of who owns what that will compel the assent of everyone concerned. It is a way of making and preserving truths" (The Economist Staff, 2015).

Originally developed to underpin the bitcoin cryptocurrency network, the blockchain has many enthusiastic supporters who see its potential beyond cash and currency (Boroujerdi & Wolf, 2015). The potential for blockchain to enable a distributed ledger of digital assets is the source of their enthusiasm (Tapscott & Tapscott, 2016: 7):

> *Some scholars have argued that the invention of double-entry bookkeeping enabled the rise of capitalism and the nation-state. This new digital ledger of economic transactions can be programmed to record virtually everything of value and importance to humankind: birth and death certificates, marriage licenses, deeds and titles of ownership, educational degrees, financial accounts, medical procedures, insurance claims, votes, provenance of food, and anything else that can be expressed in code.*

A more circumspect perspective on the potential for blockchain views the following as "genuine" blockchain use cases: (1) interorganizational recordkeeping; (2) lightweight financial systems, such as crowdfunding, gift cards, and loyalty points; (3) multiparty aggregation to address the infrastructure difficulty of combining information from large number of sources; and (4) provenance tracking (Greenspan, 2016). As it is explicated by both this and Tapscotts' perspectives, it seems that provenance tracking along a supply chain could be one of the killer apps of blockchain. Already, there are startups like provenance.org and Skuchain that are exploring this possibility. We believe that works from the computational ontology research community can be useful for these startups and other researchers interested in this topic. That is, *specifically*, we believe ontologies can contribute to develop blockchain applications for supply-chain provenance. In fact, *in general*, we believe that ontologies can contribute to developing blockchain applications.

2.1 | Why use ontologies for blockchain development?

For the general case, recall this: blockchain is a distributed database, one that is uniformly replicated across all nodes over, nearly always, a cloud computing architecture. So, not all distributed databases are consistent with a blockchain design. And blockchain is an application that is consistent with, but only one of many, applications that uses a cloud computing architecture. In order to understand data in a database distributed across numerous organizations, there must be common interpretation of data across these organizations. This interpretation can be informally enforced via use of common data standards (i.e., models, dictionaries, and conventions) and via business practices and processes that support adoption of data standards by human developers working at these organizations. Interpretation can also be formally enforced via formal specifications that enable automated inference and verification within software applications that execute on a network that spans these organizations.

Concomitantly, the classic definition of a computational ontology (Gruber, 1993) is that it is "an explicit specification of a conceptualization." In ontology-based enterprise modeling, the conceptualization is the set of ontologies required to ensure common interpretation of data from one or more enterprises' shared databases. Such ontologies can be informal or lightweight (e.g., North American Industry Classification System); formal, like TOVE Ontologies (Fox & Gruninger, 1998); or be somewhere in between (i.e., semi-formal). Making the reasonable assumption that blockchain modeling is a specialized form of interenterprise modeling, we make the case that ontology-based blockchain modeling will result in a blockchain with enhanced interpretability. That is:

- A modeling approach based on informal or semi-formal ontologies can lead to better data standards, facilitate search and understanding (especially in a supply-chain context), and business practices and processes for developing and operating a blockchain.
- A modeling approach based on formal ontologies can aid in the formal specifications for automated inference and verification in the operation of a blockchain.

It is this latter point that is particularly interesting, because that description is very similar to the definition of *smart contracts* as "pieces of software that represent a business arrangement and execute themselves automatically under pre-determined circumstances" (The Economist Staff, 2016). Smart contracts are critical for widespread blockchain adoption. Arguably, the second and third largest blockchain endeavors are Ethereum and R3CEV, with the first clearly being bitcoin. Ethereum is a worldwide platform for implementing distributed applications. It is run on a public, permission-less blockchain upon which smart contracts are executed. Ethers represent Ethereum's crypto-currency paid to participants who maintain the blockchain; there are USD 26.2 billion equivalent of ethers in

circulation. R3CEV is a startup funded for USD 200 million by a consortium of over 40 financial institutions worldwide. Its main focus has been to develop smart contracts that access block-chained distributed ledgers of consortium institutions to, for example, automatically execute terms of interest rate swaps between two banks (Rizzo, 2016).

Given the importance placed on smart contracts, the following modified statement outlines a very compelling rationale for ontology-based blockchains:

- A modeling approach based on formal ontologies can aid in the formal specifications for automated inference and verification in the operation of a blockchain. *That is, a modeling approach based on formal ontologies can aid in the development of smart contracts that execute on the blockchain.*

Now that we have a made a general case for developing ontology-based blockchains, we make the specific case: In the next section, we outline the TOVE Traceability Ontology as an apt source for our ontology-based blockchain for supply-chain provenance, and present excerpts relevant to develop a proof-of-concept.

3 | TRACEABILITY-ONTOLOGY-BASED BLOCKCHAIN FOR PROVENANCE

As mentioned, blockchain startups like provenance.org and Skuchain are working on supply-chain provenance. However, there do not appear to be any studies other than our own effort at taking an ontology-based blockchain approach. There have been some related studies: there are ontologies for supply chains (Frey, Woelk, Stockheim, & Zimmermann, 2003; Grubic & Fan, 2010; Smirnov & Chandra, 2000), though not for traceability; and traceability work on the "Supply Chain of Things," though not using ontologies (O'Leary, 2008; Ringsberg, 2011), and more interestingly, on configuring blockchain architectures for supply chain (O'Leary, 2017). A noteworthy effort develops the EAGLET ontology for ensuring data interoperability between diverse IoT devices over a supply chain (Geerts & O'Leary, 2014).

Even more expansive than this effort in terms of the focus on traceability is the one (Kim, Fox, & Gruninger, 1995) that served as a key part of TOVE ontologies for enterprise modeling (Fox & Gruninger, 1998). EAGLET and REA (Gailly & Poels, 2007) ontologies, for example, are more modern and hence incorporate the state of the art in ontology development literature. However, these still do not have the breadth or formality of traceability concepts found in the TOVE Ontology: the TOVE Traceability Ontology represents broad traceability concepts as built up from core enterprise modeling constructs, is conceptually represented in first-order logic, and implemented in Prolog. TOVE's traceability work has garnered interest, interestingly, from food sciences researchers (Dabbene, Gay, & Tortia, 2014; Regattieri, Gamberi, & Manzini, 2007). Food science has co-opted what was an ontology biased toward manufacturing industry enterprise modeling to ensure food safety along the food supply chain. This bodes well for using the TOVE Traceability Ontology as the primary source to design our blockchain.

3.1 | TOVE traceability ontology excerpt

The key informal assumptions used in developing this ontology are as follows:

- It must be possible to trace from one entity to another, where neither the entities are abstracted entities.
- *Traceable resource unit* (*TRU*—a representation for a batch of a something; e.g., a tru of 100 widgets) is the resource representation that must be traceable, since a tru is neither an abstracted nor an aggregated entity.
- Primitive activity is the activity representation that must be traceable, since a primitive activity is neither an abstracted nor an aggregated entity.

Figure 1 is a simplified version of the data model for the ontology (Kim et al., 1995), and Figure 2 shows some of the key axioms of the ontology expressed formally in first-order logic (Kim, 1999).

4 | PROOF-OF-CONCEPT IMPLEMENTATION OF ONTOLOGY-BASED BLOCKCHAIN FOR SUPPLY-CHAIN PROVENANCE

In this section we describe how we interpret the traceability ontology as a real-time tracking system, capable of tracing the provenance of TRUs back to any other TRUs in their provenance history or chain.

As suggested by Figure 3, blockchain technologies are built upon Internet technologies, using a Web browser as a natural interface. We used The Truffle framework by ConsenSys (https://consensys.net/) to generate a JavaScript-based (Web3 ABI) interface to interact with the deployed smart contract, forming inputs, or predicates, into the system to define the state of objects, as well as performing traces.

The state of the blockchain or distributed ledger in Ethereum represents the state of all deployed programs, or contracts, in terms of inputs, internal variables, and outputs (e.g., logs). All Ethereum clients on the network can participate in maintaining the ledger by listening for, computing, verifying, and encoding transactions into blocks (i.e., mining). Solidity is currently the main programming language on the Ethereum platform, and it is purpose built for writing smart-contract-style programs. Solidity is an object-oriented language, in which the `Contract` is the fundamental class for encapsulating programs or smart contracts in Solidity. While the language and platform are representationally Turing complete—they can be used to represent any possible computation—in practice, computations within `Contracts` are subject to constraints. These are in turn due to the economic incentives used to reward the decentralized network of individuals who carry out computations on the blockchain in order to determine its next state, or block. That is, that all transactions have a cost that has to be paid in ethers, Ethereum's native token-based currency.

Data models such as Figure 1 used in ontology-based enterprise modeling and their subsequent implementation in object-oriented

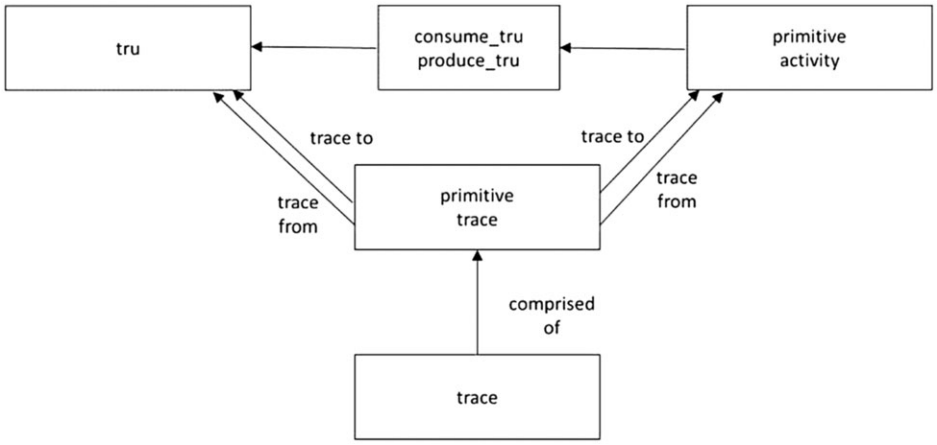

FIGURE 1 Simplified TOVE Traceability Ontology data model

Trace Axiom: Cons-1. **A tru is produced only once.**

$$\forall A \forall St_1 \forall Rt \forall s \left[holds(produce(St_1,A),s) \wedge holds(produces(St_1,Rt),s) \supset \neg \exists St_2 \{ holds(produce(St_2,A),s) \wedge holds(produces(St_2,Rt),s) \wedge St_1 \neq St_2 \} \right].$$

Rt: a tru
St_1,St_2: the same state describing the production of Rt
A: an activity which produces Rt
s: an extant situation

T4. *If a tru is used or consumed by a primitive activity, then the resource point of the tru at time T_p (the time at which the use or consume state is enabled) is the amount of the tru that has not been committed, and hence is available for other states.*

FIGURE 2 Key axioms of the TOVE Traceability Ontology

programming environments have been extensively explored in the literature (Evermann & Wand, 2005; Siricharoen, 2007). One such methodology commonly used as an intermediary representation for translating business processes into the language of object-oriented software engineering is UML (Eriksson & Penker, 2000).

Figure 4 is a pertinent diagram. As implied by Figure 4, the only public interfaces are provided by the `Trace` class; therefore, all user input affecting the contract state on the blockchain is through the `Trace` class. All output communication from the `Contract` is accomplished through the use of `Events`, which are log data variables to the blockchain and in turn read by the Ethereum client. Ensuring that constraints and relationships implied by the axioms are applied to the system is analogous to maintaining the so-called class invariants within the system of objects (Meyer, 1988).

Translating from formal ontology representations to Solidity can be problematic due to Solidity's novelty and correspondingly low maturity. Nevertheless, if ontologies are to be used programmatically, Solidity's *de facto* position as the programming language for Ethereum necessitates this difficult translation. Promisingly, there has been some recent work on designing translation mechanisms (Kim & Laskowski, 2017). Therefore, as shown in Figure 4, the `Trace` object or class is represented as a `Contract`, and the `PrimitiveActivity` and `Tru` are represented as struct types, which are essentially classes without associated methods, or functions bound to each object instance. PrimitiveActivity and Tru could also be implemented as contracts. As noted in the listings and UML diagram, `Trace` has several functions defined, which encompass the behavior and constraints upon the encapsulated types, `PrimitiveActivity` and `Tru`. The public functions comprise the public interface for the contract. The notion of a `Primitive Trace` is thereby implemented as a public function or method rather than an object or variable, which is a common approach to implementing computed fields or variables bound to objects (Meyer, 1988).

In this section we elucidate the functionality of the source code for the smart contract that implements the functionality described in Section 3. The full version is available for download at https://github.com/professormarek/traceability. The gist of the code is that we are able to record the scenario pictured in Figure 5 to the Ethereum blockchain, and the smart contract implementation of the ontology axioms is used to generate the trace shown in Figure 6.

In this section we describe how we adapted a simplified TOVE ontology (Fox & Gruninger, 1998; Kim et al., 1995) and implement the ontological enterprise model as a real-time tracking system, capable of tracing the provenance of a TRU back to any previous TRU in their provenance history or chain. Our simplified ontology includes the following axioms:

Axiom 1. A TRU is produced only once.

Axiom 2. A TRU (resource) can only be consumed if it is available (exists and is not consumed).

Axiom 3. A Primitive Activity consumes one TRU and produces one TRU.

Axiom 4. A TRU is first known to exist at the time it is first consumed or produced.

Axiom 5. Once consumed, a TRU cannot become unconsumed.

For ease of discussion the smart contract is broken up into coherent code listing. Code Listings 1–8, when assembled in order, will yield the entire `Trace` contract. Readers familiar with the JavaScript language will immediately recognize many of the syntax features of Solidity.

In code Listing 1, the `Trace` contract is declared, which will serve as our main object and interface to the Blockchain Smart Contract. The data contained within the `Trace` is also declared here.

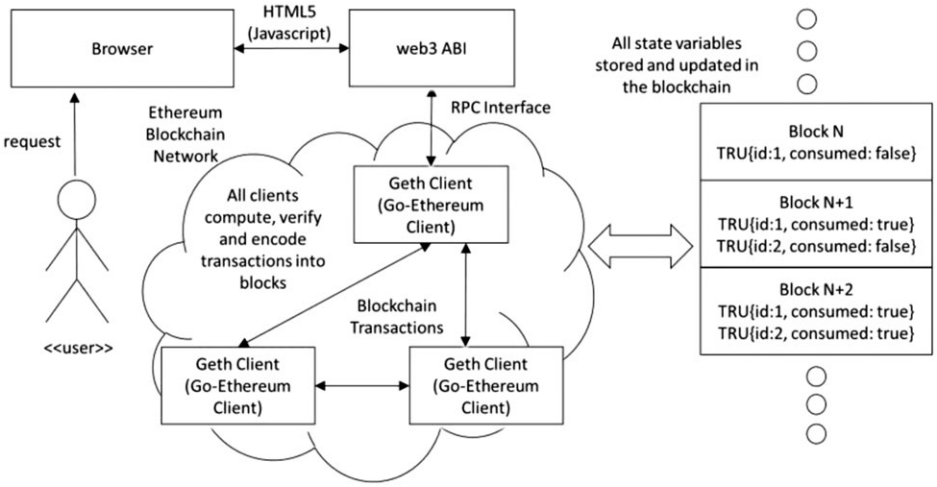

FIGURE 3 A system diagram depicting mediated user interaction with the blockchain application

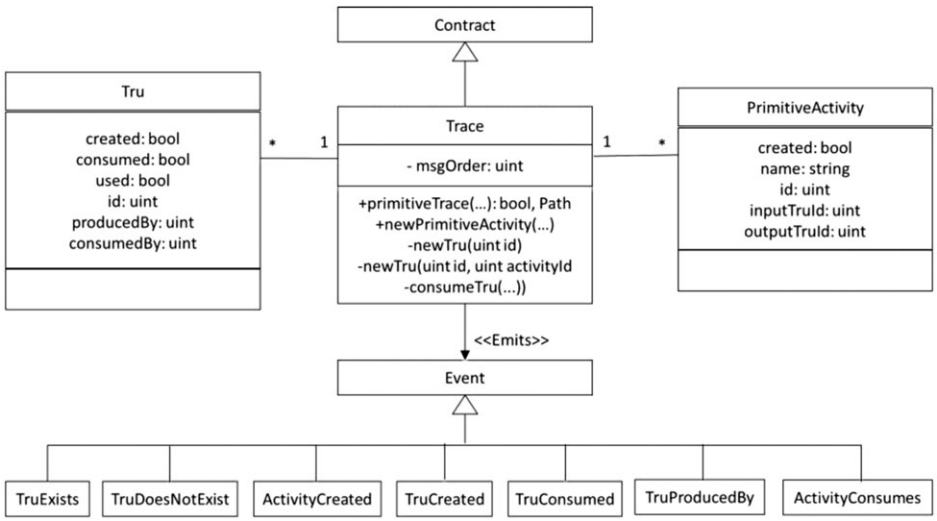

FIGURE 4 A UML diagram depicting the object-oriented design of the traceability data model, as implemented in Ethereum-–Solidity

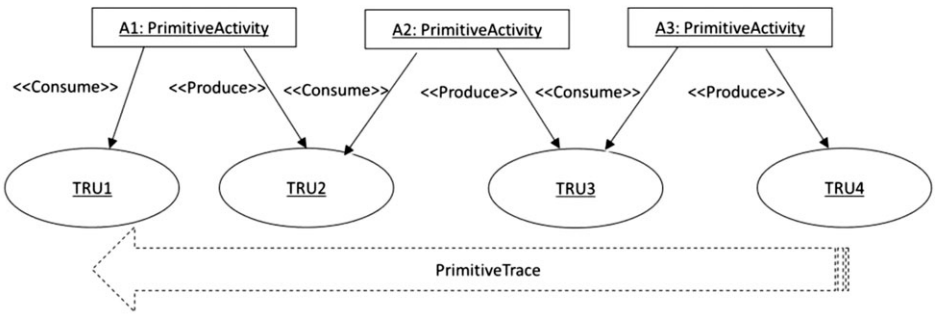

FIGURE 5 Trace scenario considered for proof-of-concept demonstration

The data types `Tru` and `PrimitiveActivity` are declared (lines 2–16), as well as lookup tables (lines 17 and 18) that will permit the Trace to look up and refer to instances of `Tru` and `PrimitiveActivity` by their respective Id. The `msgOrder` variable is there to compensate for asynchronous delivery of messages to the user interface (implemented in JavaScript) by providing an ordered index for each message delivered to the user. The constructor function on lines 20–22 is automatically called when the contract is first deployed, and it sets the message index to zero.

Listing 1 Definition of Trace, internal structures, data members, and constructor

```
1   contract Trace{
2     struct Tru{
3     bool consumed;
4     bool used;
5     bool created;
6     uint id;
```

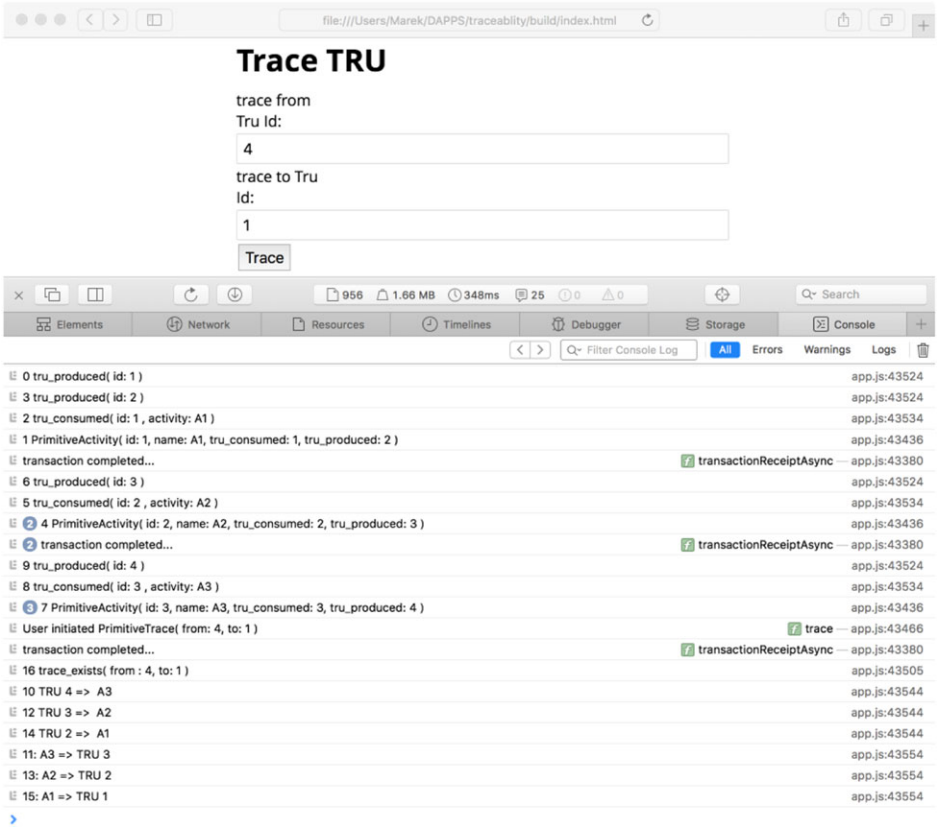

FIGURE 6 Screen output of a trace executed on the Ethereum blockchain

```
7     uint producedBy;
8     uint consumedBy;
9   }
10  struct PrimitiveActivity{
11    bool created;
12    string name;
13    uint id;
14    uint inputTruId;
15     uint outputTruId;
16}
17  mapping(uint => Tru) truLookup;
18 mapping(uint => PrimitiveActivity) activityLookup;
19  uint msgOrder;
20    function Trace(){
21    msgOrder = 0;
22  }
```

In Listing 2, several so-called function modifiers are shown. Each can be thought of as a check for valid preconditions before executing any code. This is useful for enforcing the invariant of the class, as well as implementing the specified axioms. If execution reaches the underscore "_" character in the modifier (lines 5, 11, 17, 23, 29, and 35); execution continues with the next modifier applied to the function, or if all modifiers have been passed, then execution continues with the first line of the body of the function. Typically, the pattern is that if the condition described by the modifier is not met, the throw statement will be executed. The effect of throw, is that the execution of the transaction halts immediately, and any changes resulting from that transaction are rolled back (i.e., not recorded in the blockchain). Examples of modifiers applied to functions are found in Listings 4–8; the modifiers applied are listed following the function header (starting with line 2).

Listing 2 Function modifiers used to check preconditions before executing contract methods

```
1   modifier nonZero(uint num){
2     if(num == 0){
3       throw;
4     }
5     _
6   }
7   modifier truDoesNotExist(uint id){
8     if(truLookup[id].created){
9       throw;
10    }
11    _
12  }
13  modifier truExists(uint id){
14    if(truLookup[id].created! = true){
15      throw;
16    }
17    _
18  }
19  modifier primitiveActivityDoesNotExist(uint id){
20    if(activityLookup[id].created){
21      throw;
22    }
```

```
23     _
24  }
25   modifier primitiveActivityExists(uint id){
26     if(activityLookup[id].created! = true){
27       throw;
28     }
29     _
30  }
31    modifier truAvailable(uint id){
32     if(truLookup[id].consumed||truLookup[id].used){
33         throw;
34     }
35     _
36  }
```

Each `Event` found in Listing 3, is a signal or message that is logged to the blockchain, as appropriate, when called from the functions in Listings 4–8. The log is then read by the Geth client in Figure 3, and ultimately communicated to the user through the Web browser interface. The first argument to each is an index used to indicate the order of messages, as the JavaScript-based client receives them asynchronously by design.

Listing 3 Events used to communicate state change to observers outside of the blockchain

```
1  event TraceExists(uint msgOrder, uint from_tru, uint
to_tru);
2  event TraceDoesNotExist(uint msgOrder, uint from_tru,
uint  to_tru);
3  event ActivityCreated(uint msgOrder, uint activityId,
string  description,  uint  consumedTruId,  uint
producedTruId);
4   event TruCreated(uint msgOrder, uint truId);
5   event TruConsumed(uint msgOrder, uint truId, uint
activityId, string activityName);
6   event TruProducedBy(uint msgOrder, uint currTru,
uint currActivity, string activityName);
7   event  ActivityConsumes(uint  msgOrder,  uint
currActivity, string activityName, uint currTru);
```

The `newTru` function in Listing 4 is called whenever it is necessary to initialize or create a new `Tru`. Notice that in Listing 4, the existence of a `Tru` with the given Id is checked for by the modifier `truDoesNotExist` on line 2 before the `Tru` is created, enforcing Axiom 1, that a `Tru` can only be created once. The `TruCreated` event on line 11 is used to notify the user that a new `Tru` has been added to the system. This function is private to prevent users from violating Axiom 4 by creating instances of `Tru` that have not yet been either consumed or produced. This function is used to create a `Tru` that has not been produced by any `PrimitiveActivity` in the system, but is consumed by a `PrimitiveActivity` (see Listing 7, line 11).

Listing 4 Method or function for creating a new Tru

```
1  function newTru(uint id) private
2  truDoesNotExist(id)
3  nonZero(id)
4  {
5     truLookup[id].created = true;
6     truLookup[id].id = id;
7     truLookup[id].consumed = false;
8     truLookup[id].used = false;
9     truLookup[id].producedBy = 0;
10    truLookup[id].consumedBy = 0;
11    TruCreated(msgOrder++, id);
12  }
```

The `newTru` function in Listing 5 is used to create a `Tru` that is produced by a newly instantiated `PrimitiveActivity` (see Listing 7, line 16). As with the previous `newTru` function in Listing 4, the `newTru` function in Listing 5 prevents users from violating Axioms 1 and 4 by applying the `truDoesNotExist` modifier and being declared private respectively.

Listing 5 Method or function for creating a new Tru that is produced by a PrimitiveActivity

```
1  function newTru(uint id, uint activityId) private
2  truDoesNotExist(id)
3  nonZero(id)
4  primitiveActivityExists(activityId)
5  {
6     newTru(id);
7     truLookup[id].producedBy = activityId;
8  }
```

The `consumeTru` function in Listing 6 is used when creating a new `PrimitiveActivty` in order to consume the requisite `Tru` as implied by Axiom 3. The `truAvailable` modifier (line 3) is used to enforce Axiom 2.

Listing 6 Method or function to consume a `Tru` when consumed by a `PrimitiveActivity`.

```
1   function consumeTru(uint truId, uint activityId)
    private
2  truExists(truId)
3  truAvailable(truId)
4  primitiveActivityExists(activityId)
5  {
6     truLookup[truId].consumed = true;
7     truLookup[truId].consumedBy = activityId;
8     TruConsumed(msgOrder++,  truId,  activityId,
      activityLookup[activityId].name);
9}
```

Axiom 3 is enforced by the logic in the `newPrimitiveActivity` function in Listing 7, as the modifiers require a created `inputTru` and not-yet-created `outputTru`. This function will consume the `inputTru` and produce the `outputTru` while instantiating a new Primitive Activity with the specified Id.

Listing 7 Method or function used to create a new PrimitiveActivity

```
1    function newPrimitiveActivity(string name, uint
id, uint inputTruId, uint outputTruId)
2   truAvailable(inputTruId)
3   truDoesNotExist(outputTruId)
4   primitiveActivityDoesNotExist(id)
5   nonZero(id)
6   {
7      activityLookup[id].name = name;
8      activityLookup[id].id = id;
9      activityLookup[id].created = true;
10     if(truLookup[inputTruId].created! = true){
11        newTru(inputTruId);
12     }
13     ActivityCreated(msgOrder++, id, name, inputTruId,
       outputTruId);
14     consumeTru(inputTruId, id);
15     activityLookup[id].inputTruId = inputTruId;
16     newTru(outputTruId, id);
17     activityLookup[id].outputTruId = outputTruId;
18  }
```

The `primitiveTrace` function (Listing 8) implements the Primitive Trace functionality as a function rather than an object or class. It searches the trace history, where `tru_begin` represents a `Tru` that has been produced as a result of a `PrimitiveActivity` consuming `tru_end` at some point in the past. In effect, a `primitiveTrace` will exist from `tru_begin` to `tru_end` if and only if `tru_begin` has `tru_end` at some point in the history of `tru_begin`. To determine whether a trace exists in the other direction, the arguments can be interchanged.

Listing 8 Method or function called by the user to perform a Primitive Trace

```
1   function primitiveTrace(uint tru_begin, uint
    tru_end)
2   truExists(tru_begin)
3   truExists(tru_end)
4   {
5        uint currTru = tru_begin;
6      while(currTru!=tru_end && truLookup[currTru].
       producedBy!=0){
7        uint currActivity = truLookup[currTru].
         producedBy;
8         var activityName = activityLookup[currActivity].
          name;
9        TruProducedBy(msgOrder++, currTru, currActivity,
         activityName);
10        currTru = activityLookup[currActivity].
          inputTruId;
11       activityName = activityLookup[currActivity].
         name;
12       ActivityConsumes(msgOrder++, currActivity,
          activityName, currTru);
13    }
14    if(currTru == tru_end){
15      TraceExists(msgOrder++, tru_begin, tru_end);
16    }
17    else
18    {
19      TraceDoesNotExist(msgOrder++,tru_begin, tru_
        end);
20    }
21  } //end of function primitiveTrace
22  } //end of contract Trace
```

Axiom 5 is enforced by the fact that we see from Listings 1–8 that there is no function that can be called externally in order to reset a consumed `Tru` back to an unconsumed state, that thanks to encapsulation there is no API functionally to permit the resetting of a consumed `Tru` back to an unconsumed state.

5 | CONCLUDING REMARKS AND FUTURE WORK

We identified evaluating provenance as an important and ongoing business issue. Evaluating knowledge provenance has become more possible as more and more of the data required to discover the source of knowledge is recorded on the Web. Evaluating provenance of physical goods—or what we call supply chain provenance—has generally been more difficult because so many goods are handled in complex, international supply chains where granular tracking of physical characteristics and product whereabouts has not been possible. That is, until recently, when provenance evaluation has become more possible with the advent of IoT and blockchain.

In particular, as blockchain technology evolves, as more business models that leverage it are conceived, and as more researchers explore still-nascent research opportunities with its use, we believe that the ontological engineering community can make a contribution to the growth of blockchain. Even though ontologies can be used in applications more broad than just blockchain, this new technology represents an interesting and potentially important application area for ontologies. We posit one specific and two general potential contributions and present preliminary results in this paper as a proof of concept of these contributions.

- Specifically, ontologies that represent fundamental concepts in traceability can contribute domain knowledge to develop blockchain applications for supply-chain provenance. As a proof of concept, we wrote source code on the Ethereum blockchain and assessed that we could in fact program concepts from the TOVE Traceability Ontology in a blockchain platform.
- Generally, a modeling approach based on informal or semi-formal ontologies can lead to better data standards, and business practices and processes for developing and operating a blockchain. As a proof of concept, we analyzed excerpted assumptions and

data models of the TOVE Traceability Ontology and used them to develop the appropriate distributed ledger on the blockchain.

- Generally, a modeling approach based on formal ontologies can aid in the development of smart contracts that execute on the blockchain. As a proof of concept, we translated TOVE Traceability Ontology axioms that were expressed in first-order logic into smart contracts that could execute a provenance trace and enforce traceability constraints on the blockchain.

There is much more work to be done. Specifically, there are many more traceability constructs—both informal data models and formal axioms—that ought to be encoded to enhance blockchain provenance capabilities. Generally, more research is needed to make the conversion from ontology representations to blockchain code more systematic. That may entail more granularly outlining conversion steps, developing custom APIs, or contributing to efforts to convert semantic Web representations like OWL and RDF into blockchain-compliant representations.

Future work notwithstanding, we believe that we have already made some contribution toward providing guidance for those wishing to use ontologies to develop blockchain applications, and more specifically for evaluating supply-chain provenance.

ENDNOTES

[1] According to coindesk.com, as of July 3, 2017. In contrast, there are USD 42.6 billion equivalent bitcoins in circulation.

[2] https://www.cryptocoinsnews.com/report-blockchain-r3-seeks-200-millionbackers/.

[3] Figure 6 refers to "transactions completed." This is an expression used in the Solidity/Ethereum output generator to denote a processing step in the environment, so it should not be confused with the ontology term, activity.

REFERENCES

Armstrong S. 2016. Eight things you need to know about the future of retail. *Wired UK*. http://www.wired.co.uk/article/wired-retail-2015 (accessed February 27, 2018).

Boroujerdi RD, Wolf C. 2015. Emerging Theme Radar: What if I told you…. *Goldman Sachs Equity Research*, December 2. http://www.goldmansachs.com/our-thinking/pages/macroeconomic-insights-folder/what-if-i-told-you/report.pdf (accessed February 27, 2018).

Dabbene, F., Gay, P., & Tortia, C. (2014). Traceability issues in food supply chain management: a review. *Biosystems Engineering*, *120*, 65–80.

Erickson, J. S., Sheehan, J., Bennett, K. P., & McGuinness, D. L. (2016). Addressing scientific rigor in data analytics using semantic workflows. In M. Mattoso, & B. Glavic (Eds.), *Provenance and Annotation of Data and Processes. IPAW 2016* (Vol. 9672)*Lecture Notes in Computer Science* (pp. 187–190). Cham: Springer.

Eriksson, H.-E., & Penker, M. (2000). *Business Modeling with UML: Business Patterns at Work*. New York: John Wiley and Sons, Inc.

Evermann, J., & Wand, Y. (2005). Ontology based object-oriented domain modelling: fundamental concepts. *Requirements Engineering*, *10*(2), 146–160.

Fox, M. S., & Gruninger, M. (1998). Enterprise modelling. *AI Magazine*, *19*(3), 109–121.

Fox, M. S., & Huang, J. (2005). Knowledge provenance in enterprise information. *International Journal of Production Research*, *43*(20), 4471–4492.

Frey D, Woelk P-O, Stockheim T, Zimmermann R. 2003. Integrated multi-agent-based supply chain management. In *Proceedings. Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises. WET ICE 2003*. IEEE Computer Society: Los Alamitos, CA; 24–29.

Gailly, F., & Poels, G. (2007). Ontology-driven business modelling: improving the conceptual representation of the REA ontology. In C. Parent, K. D. Schewe, V. C. Storey, & B. Thalheim (Eds.), *Conceptual Modeling—ER 2007* (Vol. 4801)*Lecture Notes in Computer Science* (pp. 407–422). Berlin: Springer.

Geerts, G. L., & O'Leary, D. E. (2014). A supply chain of things: the EAGLET ontology for highly visible supply chains. *Decision Support Systems*, *63*, 3–22.

Greenspan G. 2016. Four genuine blockchain use cases—CoinDesk. *CoinDesk*, May 11. http://www.coindesk.com/four-genuine-blockchain-use-cases/ (accessed February 27, 2018).

Gruber, T. R. (1993). A translational approach to portable ontology specifications. *Knowledge Acquisition*, *5*(2), 199–220.

Grubic, T., & Fan, I.-S. (2010). Supply chain ontology: review, analysis and synthesis. *Computers in Industry*, *61*, 776–786.

Kim H, Laskowski M. 2017. A perspective on blockchain smart contracts: reducing uncertainty and complexity in value exchange. In *2017 26th International Conference on Computer Communication and Networks (ICCCN)*. IEEE Press; 1–6.

Kim HM. 1999. Representing and reasoning about quality using enterprise models. PhD thesis, University of Toronto.

Kim HM, Fox MS, Gruninger M. 1995. An ontology of quality for enterprise modelling. In *Proceedings 4th IEEE Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE '95)*, IEEE Computer Society Press: Los Alamitos, CA; 105–116.

Merriam-Webster. 2016. Provenance. http://www.merriam-webster.com/dictionary/provenance; (accessed February 27, 2018).

Meyer, B. (1988). *Object-Oriented Software Construction* (Vol. 2). New York: Prentice Hall.

Morris DZ. 2016. Leaderless, blockchain-based venture capital fund raises $100 million, and counting. *Fortune*, May 15. http://fortune.com/2016/05/15/leaderless-blockchain-vc-fund/ (accessed February 27, 2018).

Nakamoto S. 2008. Bitcoin: a peer-to-peer electronic cash system. Bitcoin.org. https://bitcoin.org/bitcoin.pdf (accessed February 27, 2018).

O'Leary, D. E. (2008). Supporting decisions in real-time enterprises: autonomic supply chain systems. *Information Systems and e-Business Management*, *6*(3), 239–255.

O'Leary, D. E. (2017). Configuring blockchain architectures for transaction information in blockchain consortiums: the case of accounting and supply chain systems. *Intelligent Systems in Accounting, Finance and Management*, *24*(4), 138–147.

Popper N. 2016. A venture fund with plenty of virtual capital, but no capitalist. *New York Times*, May 21. http://www.nytimes.com/2016/05/22/business/dealbook/crypto-ether-bitcoin-currency.html?_r=1 (accessed February 27, 2018).

Regattieri, A., Gamberi, M., & Manzini, R. (2007). Traceability of food products: general framework and experimental evidence. *Journal of Food Engineering*, *81*(2), 347–356.

Ringsberg, H. (2011). *Traceability in Food Supply Chains: Exploring Governmental Authority and Industrial Effects*. Lund, Sweden: Lund University.

Rizzo P. 2016. How Barclays used R3's tech to build a smart contracts prototype. *CoinDesk*, April 26. http://www.coindesk.com/barclays-smart-contracts-templates-demo-r3-corda/ (accessed February 27, 2018).

Siricharoen, W. V. (2007). Ontologies and object models in object oriented software engineering. *IAENG International Journal of Computer Science*, *33*(1), 19–24.

Smirnov AV, Chandra C. 2000. Ontology-based knowledge management for co-operative supply chain configuration. In *Proceedings of the 2000 AAAI Spring Symposium on Bringing Knowledge to Business Processes*. AAAI Press; 85–92. https://ocs.aaai.org/Papers/Symposia/Spring/2000/SS-00-03/SS00-03-013.pdf (accessed February 27, 2018).

Tapscott, D., & Tapscott, A. (2016). *Blockchain Revolution: How the Technology Behind Bitcoin is Changing Money, Business, and the World*. Toronto: Penguin Canada.

The Economist Staff (2015). The great chain of being sure about things. *The Economist*, (October 31). https://www.economist.com/news/briefing/21677228-technology-behind-bitcoin-lets-people-who-do-not-know-or-trust-each-other-build-dependable (accessed February 27, 2018

The Economist Staff. 2016. Not-so-clever contracts. *The Economist*, July 28. http://www.economist.com/news/business/21702758-time-being-least-human-judgment-still-better-bet-cold-hearted?fsrc=scn/tw/te/pe/ed/notsoclevercontracts (accessed February 27, 2018).

How to cite this article: Kim HM, Laskowski M. Toward an ontology-driven blockchain design for supply-chain provenance. *Intell Sys Acc Fin Mgmt*. 2018;25:18–27. https://doi.org/10.1002/isaf.1424

APPENDIX

PARTIAL LISTING OF THE TRACE CONSTRUCT

The full version is available for download at https://github.com/professormarek/traceability.

```
contract Trace{
  struct Tru{
    bool consumed;
    bool used;
    bool created;
    uint id;
    uint producedBy;
    uint consumedBy;
  }
  struct PrimitiveActivity{
    bool created;
    string name;
    uint id;
    uint inputTruId;
    uint outputTruId;
  }
  mapping(uint => Tru) truLookup;
  mapping(uint => PrimitiveActivity) activityLookup;
  uint msgOrder;
  function Trace(){
    msgOrder = 0;
  }
modifier nonZero(uint num){
  if(num == 0){
    throw;
  }
  _
}
modifier truDoesNotExist(uint id){
  if(truLookup[id].created){
    throw;
  }
  _
}
modifier truAvailable(uint id){
 if(truLookup[id].consumed||truLookup[id].used){
    throw;
  }
  _
}
modifier truExists(uint id){
  if(truLookup[id].created! = true){
    throw;
  }
  _
}
modifier primitiveActivityExists(uint id){
  if(activityLookup[id].created! = true){
    throw;
  }
  _
}
function newTru(uint id) private
truDoesNotExist(id)
nonZero(id)
{
  truLookup[id].created = true;
  truLookup[id].id = id;
  truLookup[id].consumed = false;
  truLookup[id].used = false;
  truLookup[id].producedBy = 0;
  truLookup[id].consumedBy = 0;
  TruCreated(msgOrder++, id);
}
function newTru(uint id, uint activityId) private
truDoesNotExist(id)
nonZero(id)
primitiveActivityExists(activityId)
{
  newTru(id);
  truLookup[id].producedBy = activityId;
}
function consumeTru(uint truId, uint activityId)
 private
truExists(truId)
truAvailable(truId)
primitiveActivityExists(activityId)
{
  truLookup[truId].consumed = true;
  truLookup[truId].consumedBy = activityId;
  TruConsumed(msgOrder++,  truId,  activityId,
  activityLookup[activityId].name);
}
```