Linked Data Indexing of Distributed Ledgers

Allan Third
Knowledge Media Institute
The Open University
Milton Keynes, MK7 6AA
allan.third@open.ac.uk

John Domingue
Knowledge Media Institute
The Open University
Milton Keynes, MK7 6AA
john.domingue@open.ac.uk

ABSTRACT

Searching for information in distributed ledgers is currently not an easy task, as information relating to an entity may be scattered throughout the ledger with no index. As distributed ledger technologies become more established, they will increasingly be used to represent real world transactions involving many parties and the search requirements will grow. An index providing the ability to search using domain specific terms across multiple ledgers will greatly enhance to power, usability and scope of these systems.

We have implemented a semantic index to the Ethereum blockchain platform, to expose distributed ledger data as Linked Data. As well as indexing block- and transaction-level data according to the BLONDiE ontology, we have mapped smart contracts to the Minimal Service Model ontology, to take the first steps towards connecting smart contracts with Semantic Web Services.

Keywords

Linked Data, semantic indexing, blockchains, distributed ledgers

1. INTRODUCTION

Distributed ledgers, based on blockchains [1] (described in 2), have been gaining significant attention in recent years, and for a highly-diverse set of use cases. The attributes which make a blockchain useful for underpinning *cryptocurrencies* – such as their distributed nature, proof by consensus and secure transaction recording – also make them useful in other contexts. [12] outlines a set of criteria to determine when a blockchain may be useful for an application scenario. Currently, the use of blockchains is being explored in contexts as diverse as education, supply chain management, recruitment, and so on. Ledgers such as Ethereum [26] extend the initial Bitcoin blockchain [16] with features such as *smart contracts*, which enable code to be distributed and executed on the blockchain in a trusted way, extending the possible uses of this technology even further.

WWW'17 Companion, April 3–7, 2017, Perth, Australia.
ACM 978-1-4503-4914-7/17/04.
http://dx.doi.org/10.1145/3041021.3053895

As distributed ledgers become more widely used for more diverse forms of data, the need for efficient querying of such data becomes more important, giving rise to the need for the *indexing* of ledger entries. More importantly, there will be increasing requirements to *integrate* distributed-ledger data with other forms of data and for ledgers to work well with existing technology stacks. Data stored (or hashed on) distributed ledger can relate to all kinds of content, meaning and use, and there are needs to integrate these data with arbitrary and diverse external data sources and to integrate smart contracts with services available on the Web– in other words, there is a need for Linked Data. This paper presents our initial work on creating a Linked Data index for distributed ledgers, in order to support efficient access to data and smart contracts stored on Ethereum blockchains via the Semantic Web technology stack.

2. DISTRIBUTED LEDGERS AND INDEXING

Distributed ledgers based on blockchains do not have a central registry and, due to their structure, are not straightforward to search. Blockchains are organised into *blocks*, which contain lists of ledger entries (*transactions*). Blocks are organised into what is essentially a linked list structure (the *blockchain*), a copy of which is held by every node on the blockchain network. There is no central control. Blocks are ordered by time. The initial element in the blockchain structure is known as the *genesis block* and is manually created; all subsequent blocks are added to the blockchain by a process of *consensus* between nodes, which compete to be accepted as having the network's permission to add new blocks. The specific consensus method can vary between different blockchain systems – methods can include mechanisms such as "proof of work", "proof of stake", and "proof of elapsed time" – but the overall design ensures that a block may be considered a trustworthy record of events provided the network is suitably diverse (i.e., provided that no entity controls more than 50% of all nodes).

The notion of *account* is also important. An account corresponds to an agent (a human user, perhaps) and has a notion of a *balance* in the cryptocurrency associated with the blockchain in question (e.g., Bitcoin, Ether). Accounts may *spend* or *receive* cryptocurrency, with any such transfer of value forming part of a transaction.

Systems such as Ethereum go beyond simply containing lists of transactions. As mentioned earlier, an important innovation is the idea of the "smart contract", which is a chunk of code which is stored and executed on the blockchain.

Smart contracts make it possible to have automated control of what happens with data and cryptocurrency on the blockchain with native access to all of its capabilities without involving untrusted external systems. Each Ethereum smart contract has a corresponding account, of a special "contract account" type, and can store and update data and create transactions itself. Smart contracts can serve essentially as functions, with inputs and outputs, the latter of which in particular are written to the blockchain on execution, and are therefore subject to the same verifiability as other blockchain transactions.

As indicated above, the key point to note is that blockchains are strictly time-ordered structures. Where related data exists across multiple blocks (as inevitably it must), there is no inherent way to identify, group or query it. Thus it is necessary to develop an index. By indexing smart contracts themselves, where present, as well as both ordinary and contract-created transactions, we gain the ability to search and analyse the services available on a distributed ledger, and potentially to expose them to the outside world for interoperability.

There are different levels of granularity at which indexing can be carried out. At a low level, it is necessary to index the basic entities of the distributed ledger – blocks, transactions and accounts, particularly. Aggregating data stored across the blockchain requires the ability to locate and retrieve it, and so an index at this basic level is fundamental to any higher-level querying of the ledger.

At a higher, more functional level, smart contracts – more specifically, their functional interfaces – embody a lot of the extra functionality on ledger platforms such as Ethereum, and the ability to discover and access smart contracts as computational services is useful.

Smart contracts can also be indexed according to their real-world applications. The current work, for example, is taking place in the context of our ongoing experiments in placing educational data on the blockchain. Details of these experiments can be seen at [13]; the most notable and relevant to the current work is the use of smart contracts to represent Open Badges [11], where records relating to students' achievements studying with the Open University's OpenLearn platform [17] are stored in our private ledger. Indexing the contracts, accounts and data relating to their external semantics as educational records significantly increases our ability to make use of this data, and to connect it to relevant external sources.

3. VOCABULARIES

3.1 Vocabularies for distributed ledgers

To generate interoperable Linked Data, it would be helpful to use a standard ontology or vocabulary to represent blockchain concepts.The intersection between the Semantic Web and distributed ledgers is still in its infancy, and there are as yet no widely-established ontologies or vocabularies for describing these concepts. Proposed systems and vocabularies which specify or implicitly define such a vocabulary include FlexLedger [23], EthOn [19], and BLONDiE [5].

FlexLedger specifies an HTTP API designed to wrap interactions with different types of blockchain in a common Web-accessible format. Interfaces representing ledger creation, querying, appending, and so on, are defined, with a data model described using JSON-LD [22], from which it is possible to extract a vocabulary to describe this data model. However, the vocabulary itself is not explicitly defined nor given concrete semantics.

EthOn is an OWL [2] ontology designed to describe the Ethereum blockchain. It describes classes such as block, account, message, network and state, as well as more specific types such as "contract account" (each smart contract on the Ethereum blockchain has an associated account) and relations such as "has parent block". While actively developed, EthOn is, at the time of writing, at a very early stage. It has been proposed that EthOn in time be integrated with the BLONDiE ontology.

BLONDiE (Blockchain Ontology with Dynamic Extensibility) is another OWL ontology designed to describe blockchain concepts, but unlike EthOn, it is intended to be generic rather than tied to one particular type of blockchain. For example, while BLONDiE, like EthOn, contains terms for "block", "transaction", different types of account ("balance account", "contract account") and terms for specific attributes of each of these types, such as "transaction payload" and "miner address", have been defined. It also defines specialisations of those concepts for particular blockchain implementations – for example, "BitcoinBlockHeader" and "EthereumBlockHeader" are defined differently while both remaining subclasses of "BlockHeader". It is therefore possible to use BLONDiE to link data describing transactions in different blockchains together.

Of the existing candidates, BLONDiE is the most developed vocabulary for representing blockchain concepts, with the most potential to enable reusable modelling across different distributed ledgers in the future. We therefore chose to use it in building our Linked Data index.

3.2 Vocabularies for smart contracts

While both EthOn and BLONDiE contain terms relating to the concept of a smart contract, these terms are generic in both ontologies, covering only how a contract relates to other blockchain concepts. Given that smart contracts themselves are essentially executable software, it makes sense to represent their semantics using vocabularies which are already defined for other forms of software. In particular, there is a wealth of existing work on the semantic annotation of Web services and HTTP APIs– see, for example, [10] – which might be fruitfully adapted to annotate smart contracts on a distributed ledger too. Of course, such contracts are not in fact Web APIs, and the underlying implementation technology is quite different, but the core concepts do not differ essentially. Both Web APIs and smart contracts can be seen as executable functionality exposed in a distributed environment for arbitrary (suitably authorised) third-parties to call. It seems likely that the vocabularies used to annotate Web services should be usable to annotate smart contracts too. If so, there are potential practical benefits in situations involving a mix of distributed ledgers with smart contracts and existing Web services, which we anticipate becoming common. Here, we use only the Minimal Service Model (MSM) [14], a very lightweight ontology for describing Web services, including lightweight HTTP APIs. As a "least commitment" model compatible with the available SWS standards – that is to say, being designed only represent the minimum necessary aspects of a service, the MSM does not require any Web-specific concepts which may not apply to smart contracts. Figure 2 shows the MSM, which describes services

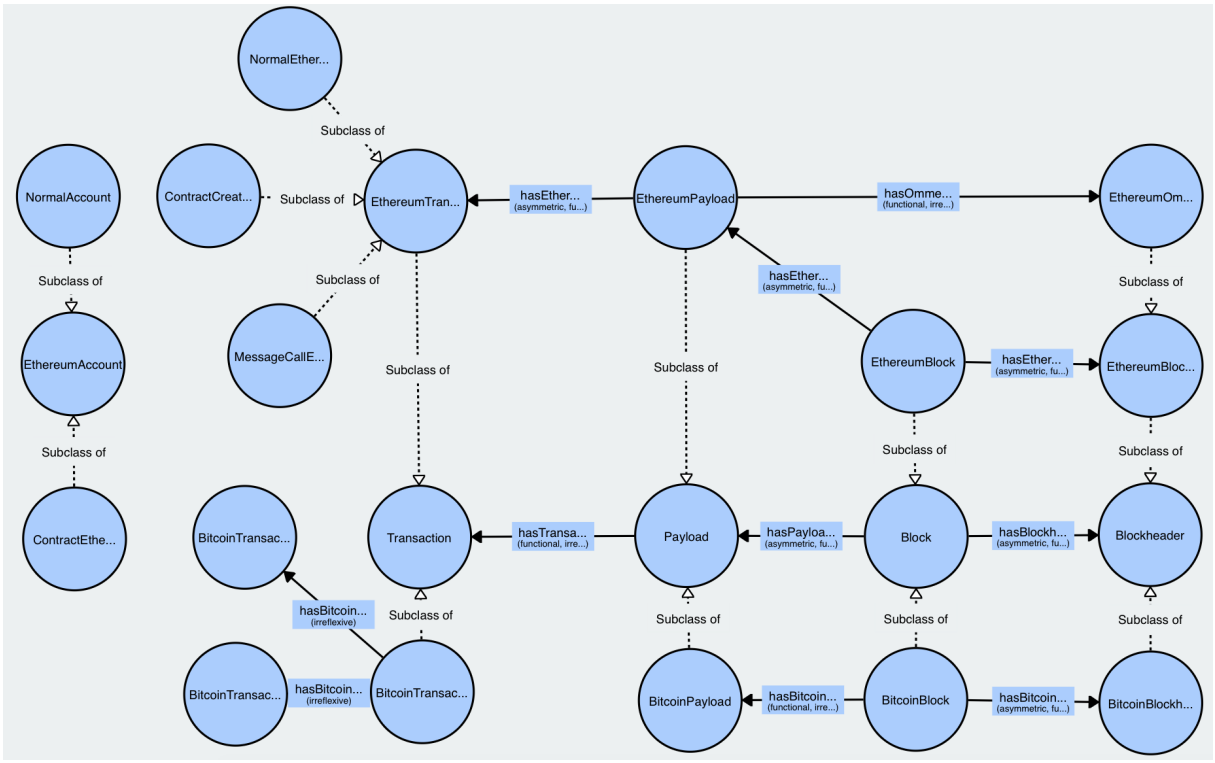

Figure 1: Visualisation of the BLONDiE ontology (created with WebVOWL [15])

as collections of operations, with inputs, outputs and faults being received or emitted as messages.

4. IMPLEMENTATION

The Linked Data index is implemented currently for the Ethereum platform. Our initial motivation was to index the data (including smart contracts) and transactions relating to our ongoing work with the educational Open Blockchain [21], which is implemented on Ethereum. This work is taking place using a private blockchain, which additionally gives the advantage of being a manageable size for experiments, by contrast to the main public Ethereum blockchain which (at the time of writing) contains over 3 million blocks [6] occupying approximately 11Gb.

Our private blockchain network currently consists of 6 nodes (5 mining nodes and one "observation" node set up for the index) which has been running since 28th of April, 2016 and with a current chain size of 2.8Gb, representing 1180460 blocks. The smart contracts deployed on this blockchain represent our experiments and ongoing development in the representation of educational achievements on the blockchain.

The Go Ethereum implementation [7] provides a Javascript [4] programming interface in the form of the `web3` library [9], which provides access to the block-level structure of Ethereum. In particular, it is straightforward to develop "listeners" in `web3` – chunks of code which are executed when new blocks are added to the blockchain – and also to iterate over every block in the chain from the genesis block forward.

The nature of a blockchain is that "older" blocks are considered to be more reliable than "newer" ones, in that the network has had time to establish a consensus about which older blocks contain the correct transaction history. While it is always theoretically possible that any block (and the chain of blocks following it) can be challenged and replaced with a different block (and subsequent chain), it is generally agreed with Ethereum that blocks approximately 12 blocks prior to the current block are reliable [3]. Depending on the application scenario in mind, one can choose between only indexing "reliable" blocks (minimising rewrites to the index) or always indexing up to the latest block (or beyond: pending transactions grouped into potential blocks are available to the network before even initial consensus has been reached, and there may be a need to index these too). As the current work is primarily a proof-of-concept with the longer-term goal of stable query access to trustworthy data, we currently do not index recent or pending blocks.

For the block-level data, we build a Linked Data index one block at a time, iterating over the contents of the block and generating RDF triples [25] using the BLONDiE ontology for each relevant piece of data, e.g., block header fields (miner address, difficulty, parent block, and so on) and transaction details. Each indexed entity is given an identifying URI based on its address in the blockchain, ensuring that all generated triples relating to the same entity will use the same URI. Figure 3 shows an example of RDF describing a block, while Figure 4 shows triples describing a transaction. The generated RDF is stored in a standards-compatible quadstore (RDF4J, [20]), where it can be queried using, e.g., SPARQL [24].

Population of the index occurs in two different ways. A listener is notified each time a new block is added to the blockchain, which triggers the indexing of the most recent "reliable" block (hardcoded to 12 blocks prior to the new

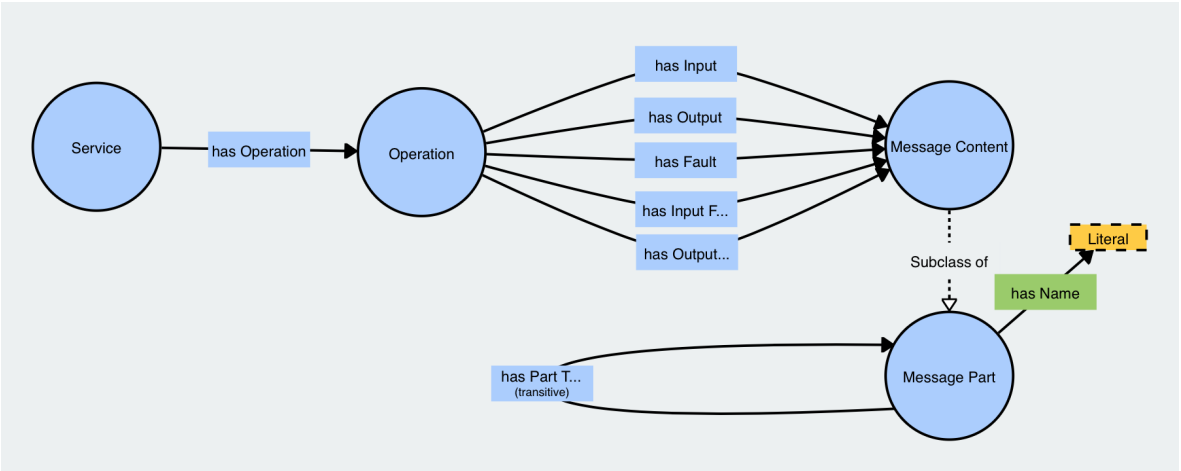

Figure 2: Visualisation of the Minimal Service Model ontology (created with WebVOWL [15])

```
kmichain:493879:0xf9752a9176e491dc85efe183c20f7fc0bebfc4e23db6e12be1edf9b6d55a8ab5 rdf:type blondie:EthereumBlock ;
       blondie:gasLimitEthereumBlock                "4712388" ;
       blondie:gasUsedEthereumBlock                 "23176" ;
       blondie:hashBlock                            "0xf9752a9176e491dc85efe183c20f7fc0bebfc4e23db6e12be1edf9b6d55a8ab5" ;
       blondie:logsBloomEthereumBlock               "0x00000000000000000000" ;
       blondie:minerAddressEthereumBlock            "0xed46d1ac16940dafee3128a4e5b41cb632aa5268" ;
       blondie:numberEthereumBlock                  "493879" ;
       blondie:sizeEthereumBlock                    "678" ;
       blondie:headerEthereumBlock                  kmichain:493879:0xf9752a9176e491dc85efe183c20f7fc0bebfc4e23db6e12be1edf9b6d55a8ab5:header ;
       blondie:containsTransactionEthereumBlock     kmichain:0x2c58a713408d8395a5a2f9360d65b806901b683a614267f6c8dca0d2b1e7262a .
```

Figure 3: RDF describing a block

```
kmichain:0x2c58a713408d8395a5a2f9360d65b806901b683a614267f6c8dca0d2b1e7262a rdf:type  blondie:EthereumTransaction ;
        blondie:blockHashEthereumTransaction               "0xf9752a9176e491dc85efe183c20f7fc0bebfc4e23db6e12be1edf9b6d55a8ab5";
        blondie:blockNumberEthereumTransaction             "493879";
        blondie:cumulativeGasUsedEthereumTransaction       "23176" ;
        blondie:gasEthereumTransaction                     "90000" ;
        blondie:gasPriceEthereumTransaction                "20000000000" ;
        blondie:gasUsedEthereumTransaction                 "23176" ;
        blondie:hashTransaction                            "0x2c58a713408d8395a5a2f9360d65b806901b683a614267f6c8dca0d2b1e7262a" ;
        blondie:inputEthereumTransaction                   "0xef65ad1ad999a485c452d2c4d0016959e0d01b8daff8c1429798a83c6bdf381d" ;
        blondie:nonceEthereumTransaction                   "29321" ;
        blondie:receivingAddressTransaction                "0x4e475617f95ce4b2e50fb52fa798a8d7356beec8" ;
        blondie:senderAddressTransaction                   "0x4e475617f95ce4b2e50fb52fa798a8d7356beec8" ;
        blondie:transactionIndexEthereumTransaction        "0" ;
        blondie:valueEthereumTransaction                   "0" ;
        blondie:containingBlockEthereumTransaction         kmichain:493879:0xf9752a9176e491dc85efe183c20f7fc0bebfc4e23db6e12be1edf9b6d55a8ab5 .
```

Figure 4: RDF describing a transaction

block on the chain), ensuring that the index remains up to date with new data. Simultaneously, a separate script iterates backwards along the chain, beginning from the block which was current at the time index generation was initiated, ensuring that historical data is also indexed.

4.1 Semantic mappings

In order to generate RDF, it is necessary to map blockchain entities to the relevant semantic terms. With block-level concepts, this is straightforward according to the definitions in BLONDiE. In order to make querying more efficient, we extend the BLONDiE schema in two ways. Firstly, records relating to both block and transactions have been augmented with a (string-valued) attribute for the *hash* of each entity, in order to provide a more direct mapping between the contents of the index and the addressable entities on the blockchain. Secondly, records relating to transactions have been augmented with links to related entities such as the containing block, the originating smart contract (if the transaction is the result of contract execution) and the input to the originating contract (again, where relevant).

The indexing of smart contracts themselves is a little more involved. The blockchain only stores a compiled, binary form of each contract, with very little metadata. In order to interact with a contract, it is necessary to have the corresponding Application Binary Interface (ABI) specification [8]. This specification is in the form of a JSON file generated when the smart contract is compiled and stored on the blockchain. Figure 5 shows an example. The ABI file specifies all functions associated with a contract, together with descriptions (names and datatypes) of the input and output parameters for each function. Given the ABI and the blockchain address of a function, it is possible to invoke or otherwise interact with the corresponding contract. In particular for the purposes of the current work, without this data it is not possible to parse contract execution logs and transaction receipts.

```
{ "badge_abi": [
    {
      "constant": false,
      "inputs": [ { "name": "imageurl", "type": "string" } ],
      "name": "changeImageURL",
      "outputs": [],
      "payable": false,
      "type": "function"
    },
    {
      "constant": false,
      "inputs": [ { "name": "tag", "type": "string" } ],
      "name": "removeTag",
      "outputs": [ { "name": "success", "type": "bool" } ],
      "payable": false,
      "type": "function"
    },
    ...
```

Figure 5: Example of a smart contract ABI

In the presence of an ABI and an address, it is straightforward to generate RDF to index the corresponding smart contract. We model each contract as an `msm:Service`, and each function as an `msm:Operation`, related (by `msm:hasInput` or `hasOutput`) as appropriate to an `msm:MessageContent` entity representing the input and output parameter names and datatypes. To retain an index into the actual blockchain contents, we augment each `msm:Service` record with a (string-valued) attribute containing the blockchain address.

Even with an ABI file, there is of course a limit to the semantic annotation available for smart contract functions. With more domain knowledge of a smart contract's purpose, it is possible to associate domain-specific terms with a contract. For example, a contract representing an Open Badge could be annotated/indexed with terms relating to the Open Badge specification and the URI identifier of the relevant course or student.

5. CONCLUSION AND FUTURE WORK

We have demonstrated the proof-of-concept for a Linked Data index onto a distributed ledger. It is now possible to query and retrieve data stored on the blockchain in disparate locations, and, more interestingly, this data can be easily linked to other sources of information using Semantic Web approaches. For example, we have been able to make blockchain smart contract functions discoverable using Semantic Web Services tools such as iServe [18]. We can also connect domain-specific data from sources external to the chain – such as linking blockchain Open Badge information with other Linked Data resources about courses.

The need for RDF entities to be identified by URIs which can be dereferenced highlights some fundamental differences between distributed ledgers and the Web. Where a Web resource can be given an identifier which contains the information needed to locate it and retrieve it, such a scheme is more difficult for blockchains. One can imagine a URI schema which, for example, identified a ledger type, e.g., Ethereum, Bitcoin, and so on, but how best to refer to the blockchain location? Identifying the hostname of a node on the relevant blockchain is possible, but does not follow the fully distributed approach, as it introduces a particular point of contact for a resource which is neither unique nor, in itself, necessary to locate that resource. If the node in question ceases to exist, the ledger and resource may persist on other nodes. By contrast, not using a node address and simply assigning a unique generated identifier for a particular blockchain leaves the URI with no concrete dereferencing information without the use of an external registry similar to, e.g., DNS.

Indexing of the Ethereum distributed ledger is limited with regard to smart contracts by the requirement to possess ABI information to interact with smart contracts. It is only possible to index contracts in detail if its author has shared the ABI – one cannot index third-party smart contracts without permission, effectively.

Of course, more remains to be done. A full evaluation is planned into performance, both raw and compared with other (non-semantic) indexing approaches to distributed ledgers. On the Linked Data side, it would be interesting to develop an HTTP wrapper to support the dereferencing of RDF URIs, as well as a layer to support the invocation of smart contracts as Semantic Web Services, and it is vital to look more deeply into the questions surrounding the URI addressing of blockchain entities.

In terms of ledger interoperability, the next step should be to extend this work to cover other blockchain platforms beyond Ethereum, enabling cross-chain semantic indexing. This is of course a large topic in its own right, with issues to be solved regarding, e.g., identity of entities between ledgers. But expressing complex relationships explicitly in a machine-readable fashion is one of the main achievements of the Semantic Web, and it is reasonable to suppose that therefore a Semantic Web approach to these issues is likely to be beneficial.

It would also be interesting to investigate the performance and cost requirements of making the index distributed in the same manner as the distributed ledger itself. The model presented here relies on an observation node, to which the index is local. While of course this can be duplicated in theory on any node where it is required, without requiring any reference to a "central" index, it is not inherently distributed and it would be useful to explore how to make it so.

As distributed ledger technologies become more established, they are likely to be used increasingly to represent real world transactions involving many parties. The ability to search using domain specific terms across multiple ledgers will greatly enhance to power, usability and scope of these systems.

We have implemented a semantic index to the Ethereum platform, to expose distributed ledger data as Linked Data. As well as indexing block- and transaction-level data according to the BLONDiE ontology, we have mapped smart contracts to the Minimal Service Model ontology, to take the first steps towards connecting smart contracts with Semantic Web Services. This work has demonstrated that such a semantic index is possible and can be useful, and has highlighted the areas which need further attention in future work.

6. REFERENCES

[1] BBC, January 2016.

[2] S. Bechhofer. Owl: Web ontology language. In *Encyclopedia of Database Systems*, pages 2008–2009. Springer, 2009.

[3] V. Buterin, 2016.

[4] E. ECMAScript, E. C. M. Association, et al. Ecmascript language specification, 2011.

[5] M. English. Blondie, 2016.

[6] Etherchain, Jan 2017.

[7] Ethereum. Go ethereum, 2017.

[8] EthereumWiki. Ethereum contract ABI, 2016.

[9] EthereumWiki. Ethereum javascript api, 2016.

[10] D. Fensel, F. M. Facca, E. Simperl, and I. Toma. *Semantic web services*. Springer Science & Business Media, 2011.

[11] E. Goligoski. Motivating the learner: Mozilla's open badges program. *Access to Knowledge: A Course Journal*, 4(1), 2012.

[12] G. Greenspan. Avoiding the pointless blockchain project, November 2015.

[13] KMi, Jan 2017.

[14] J. Kopeckỳ, K. Gomadam, and T. Vitvar. hrests: An html microformat for describing restful web services. In *Web Intelligence and Intelligent Agent Technology, 2008. WI-IAT'08. IEEE/WIC/ACM International Conference on*, volume 1, pages 619–625. IEEE, 2008.

[15] S. Lohmann, V. Link, E. Marbach, and S. Negru. WebVOWL: Web-based visualization of ontologies. In *Proceedings of EKAW 2014 Satellite Events*, volume 8982 of *LNAI*, pages 154–158. Springer, 2015.

[16] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[17] OpenLearn, 2017.

[18] C. Pedrinaci, D. Liu, M. Maleshkova, D. Lambert, J. Kopecky, and J. Domingue. iserve: a linked services publishing platform. In *CEUR workshop proceedings*, volume 596, 2010.

[19] J. Pfeffer, A. Beregszazi, C. Detrio, H. Junge, J. Chow, M. Oancea, M. Pietrzak, S. Khatchadourian, and S. Bertolo. Ethon - an ethereum ontology, 2016.

[20] RDF4J. RDF4J, 2017.

[21] M. Sharples and J. Domingue. The blockchain and kudos: A distributed system for educational record, reputation and reward. In *European Conference on Technology Enhanced Learning*, pages 490–496. Springer, 2016.

[22] M. Sporny, G. Kellogg, M. Lanthaler, W. R. W. Group, et al. Json-ld 1.0: a json-based serialization for linked data. *W3C Recommendation*, 16, 2014.

[23] M. Sporny and D. Longley. Flex ledger 1.0. W3C Blockchain Community Group, 2016.

[24] W3C. SPARQL, 2008.

[25] W3C. Resource Description Framework, 2014.

[26] G. Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, 2014.