GraphChain: A Distributed Database with Explicit Semantics and Chained RDF Graphs

Conference Paper · April 2018

DOI: 10.1145/3184558.3191554

6 authors, including:

Mirek Sopek
MakoLab USA
3 PUBLICATIONS **2** CITATIONS
SEE PROFILE

Przemyslaw Gradzki
John Paul II Catholic University of Lublin
4 PUBLICATIONS **10** CITATIONS
SEE PROFILE

Rafał Trójczak
John Paul II Catholic University of Lublin
9 PUBLICATIONS **19** CITATIONS
SEE PROFILE

Robert Trypuz
John Paul II Catholic University of Lublin
59 PUBLICATIONS **216** CITATIONS
SEE PROFILE

GraphChain – A Distributed Database with Explicit Semantics and Chained RDF Graphs*

Mirek Sopek
MakoLab S.A.
Łódź, Poland
sopek@makolab.com

Przemysław Grądzki
R&D Team, DC Lublin MakoLab S.A.
Lublin, Poland
przemyslaw.gradzki@makolab.com

Witold Kosowski
R&D Team, MakoLab S.A.
Łódź, Poland
witold.kosowski@makolab.com

Dominik Kuziński
R&D Team, MakoLab S.A.
Łódź, Poland
dominik.kuzinski@makolab.com

Rafał Trójczak
R&D Team, DC Lublin MakoLab S.A.
Lublin, Poland
rafal.trojczak@makolab.com

Robert Trypuz
R&D Team, DC Lublin MakoLab S.A.,
Faculty of Philosophy KUL
Lublin, Poland
robert.trypuz@makolab.com

ABSTRACT

In this paper we present a new idea of creating a Blockchain compliant distributed database which exposes its data with explicit semantics, is easily and natively accessible, and which applies Blockchain securitization mechanisms to the RDF graph data model directly, without additional packaging or specific serialisation. Essentially, the resulting database forms the linked chain of named RDF graphs and is given a name: GraphChain. Such graphs can then be published with the help of any standard mechanisms using triplestores or as linked data objects accessible via standard web mechanisms using the HTTP protocol to make them available on the web. They can also be easily queried using techniques like SPARQL or methods typical to available RDF graphs frameworks (like rdflib, Apache Jena, RDF4J, OWL API, RDF HDT, dotnetRDF and others). The GraphChain concept comes with its own, OWL-compliant ontology that defines all the structural, invariant elements of the GraphChain and defines their basic semantics. The paper describes also a few simple, prototypical GraphChain implementations with examples created using Java, .NET/C# and JavaScript/Node.js frameworks.

CCS CONCEPTS

• **Information systems → Resource Description Framework (RDF)**; **Web Ontology Language (OWL)**; • **Computer systems organization → Distributed architectures**;

KEYWORDS

distributed database; semantic blockchain; GraphChain; linked data

ACM Reference Format:
Mirek Sopek, Przemysław Grądzki, Witold Kosowski, Dominik Kuziński, Rafał Trójczak, and Robert Trypuz. 2018. GraphChain – A Distributed Database with Explicit Semantics and Chained RDF Graphs. In *Proceedings of The 2018 Web Conference Companion (WWW'18 Compaion).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3184558.3191554

*Operational Programme Smart Growth, project number 01.01.01-00-0982/16 funded by the National Centre for Research and Development

WWW'18 Compaion, April 23–27, 2018, Lyon, France

ACM ISBN 978-1-4503-5640-4/18/04..
https://doi.org/10.1145/3184558.3191554

1 INTRODUCTION

There is no doubt that the rise of the Blockchain technology changed the way in which modern distributed databases are designed and implemented. Blockchain offers several important benefits over traditional distributed database models. The most important are cryptographically guaranteed immutability and persistence of data stored on Blockchain accompanied by total decentralisation of the database management [25]. However, almost all existing implementation of Blockchain have used simplified data structures on top of which cryptographic algorithms are implemented. In practice, developers who wanted to store structured data on the Blockchain were required to pack or embed their data into blocks. Depending on the specific implementation, the blocks were binary or could use serialised string data. In some other Blockchain implementations, the block data structure is quite complicated, yet it still cannot be efficiently used for structured data storage[1]. For example, if a developer needed to store a tabular, tree-based or directed-graph data structure on Ethereum Blockchain he or she would typically serialise the structure into JSON object and embed it into Solidity scripts. While this is always possible, such strategy makes the embedded data hard to query: the objects need first to be extracted from blocks and then copied to a storage appropriate to their type. Only then the required query can be performed. Doing that across many, or sometimes all objects in a given domain, could require large scale extraction operation that would be very time consuming and inefficient.

What is even more important, in many real-world applications, it is crucial to work directly with much more advanced data structures. For example, when Blockchain technology is to be used in digital identity applications, we need an efficient mechanism for rich representation of the identifiable object reference data to be present on the Blockchain [2]. In another application, where Blockchain could be used for non-repudiatory storage of financial reports, it is desirable

[1] https://ethereum.stackexchange.com/questions/268/ethereum-block-architecture
[2] Mirek Sopek, "Using Blockchain for Digital Identifiers: Improving Data Security and Persistence for Digital Object Identifier (DOI) and Legal Entity Identifier (LEI)". The E-Finance Lab and DZ BANK 2016 Fall Conference. Goethe University Frankfurt. September 1st, 2016.

to store the reports right on the Blockchain. However, the limitations of the technology forced the early adopters of such processes to store on the Blockchain only the metadata about the reports, leaving the reports themselves outside the realm of the protected infrastructure[3].

In the project initiated by the authors of this paper, we are building Blockchain based infrastructure for the Legal Entity Identifier (LEI) system. Initially, we built two Proof-of-Concepts using a private Ethereum Blockchain[4]. While we proved the feasibility of the approach, we also discovered important shortcomings related to the use of such kind of blockchains for large set of structured data used today by systems like the global LEI system: the first one was related to the inability to query the "blockchained" set efficiently, while the second was related to the complicated addressing scheme which made the retrieval of reference data difficult and inefficient.

These findings led us to the idea of creating an entirely new solution where the fundamental underlying data model is the collection of linked named RDF Graphs. The Blockchain mechanisms such as data hashing, signatures, linking of graphs, distributed data replication (in analogy to blocks linking on the standard Blockchain) and achieving consensus are then implemented "on top" of the stored graphs in multiple "sites" forming the distributed database of chained RDF named graphs. The fundamental benefit of this approach is that on each site its users can work with the chained named graphs using methods that are fully developed and widely used today: SPARQL for quering, Linked Data mechanisms for accessing the nodes of the graphs, frameworks like Jena for arbitrary operations, reasoning using ontologies and so on. We named this solution GraphChain.

For the current use case (the proposed infrastructure of the global LEI system[5] the proposed solution will form a private, "permissioned" system with Proof-of-Authority as the consensus mechanism, allowing, among other things, for unlimited querying over the entire the LEI system reference data set.

However, as the focus of this paper is solely on the underlying data model of the system, we deliberately left the important considerations of the consensus mechanism for the future publications on the GraphChain.

2 THE GRAPHCHAIN DEFINITION

Let us start with introducing a few definitions related to the RDF graphs.

An *RDF triple*[6] contains three components: *the subject*, which is an RDF URI reference or a blank node, *the predicate*, which is an RDF URI reference and *the object*, which is an RDF URI reference, a literal or a blank node. An *RDF graph* is an unordered set of RDF triples and a *named RDF graph* is an RDF graph which is assigned a name in the form of a URI [5].

The main idea behind GraphChain is to use Blockchain mechanisms on top of an abstract RDF graph data type. Following the general definition of the abstract data type (ADT) [6], we define the abstract RDF graph data type by its behaviour in computer systems, the set of possible values and the operations permitted on these values. Such an approach allows us to create a Blockchain-compliant system on top of *any* concrete data structure used to represent and expose the graph. In general GraphChain does not assume any specific serialisation of the RDF graph (like RDF/XML, Turtle or JSON-LD) nor a mechanism for storage (triple store, files, memory) or exposing the graph (HTTP, file read etc). GraphChain is thus defined as:

(1) A linked chain of named RDF graphs specified by the GraphChain ontology.
(2) A set of general mechanisms for calculating a digest of the named RDF graph.
(3) A set of network mechanisms that are responsible for the distribution of the named RDF graphs among the distributed peers.

The following two definitions make the idea of GraphChain more precise:

Definition 2.1 (Linked chain of named RDF graphs). A linked chain of named RDF graphs is a linearly ordered collection (a chain) of cryptographically secured named RDF graphs. The collection is ordered by a functional relation pointing to the previous graph in the chain. With exception to the first graph in the chain, the relation is asymmetric and irreflexive.

Definition 2.2 (GraphChain). A linked chain of named RDF graphs starting from the first graph called genesis unit[7] is called the GraphChain. The GraphChain acts as a consistent named graphs history on which all nodes eventually agree[8].

3 THE GRAPHCHAIN ARCHITECTURE

A single node consists of several parts: a web interface for communication with clients (via the HTTP protocol), a web socket endpoint for communication with others nodes, a cryptography module for handling of digest calculation, a triple store repository manager for storing blocks as sets of triples and obtaining blocks from the repository, and services which bind all these parts together.

In figure 1 a diagram presenting nodes interaction is depicted. Clients can connect to a node via HTTP and to a triple store via SPARQL over HTTP. The interaction with the node itself enables a client to create new blocks (a process of block creation is explained below), add new peers, and retrieve block meta-information. Interaction with the triple store enables reading data from it with the usage of the SPARQL query language. Interaction among nodes is implemented by means of the web sockets protocol. Nodes can interact with their triple stores by various ways. These ways can be determined for example on the basis of required performance. For now we use SPARQL over HTTP.

The block creation process begins with an HTTP request from the client. The HTTP request contains a serialised RDF graph (see figure 2). For now, we assume that this graph is serialised in the Turtle format, but it should be fairly easy to extend the application

[3]European Financial Transparency Gateway, SMART 2016, European Commission, DG FISMA, Tamás SZABÓ

[4]Mirek Sopek, "How Can Blockchain amplify Digital Identifiers? Improving Data Persistence, Openness, and Trust in the modern world." DataAmplified 2016 – XBRL and the future of Business Reporting, Singapore, Nov. 8-10, 2016. https://www.dataamplified.org/sessions/blockchain-smart-contracts-and-efficiency/.

[5]See for example: https://lei.info.

[6]https://www.w3.org/TR/rdf11-concepts/#section-triples

[7]Unit is a counterpart of a Blockchain's block. In section 5 we will define precisely how it is understood in our framework. Henceforward we shall use both terms "unit" and "block" as synonyms.

[8]Compare with [25, Definition 7.20].

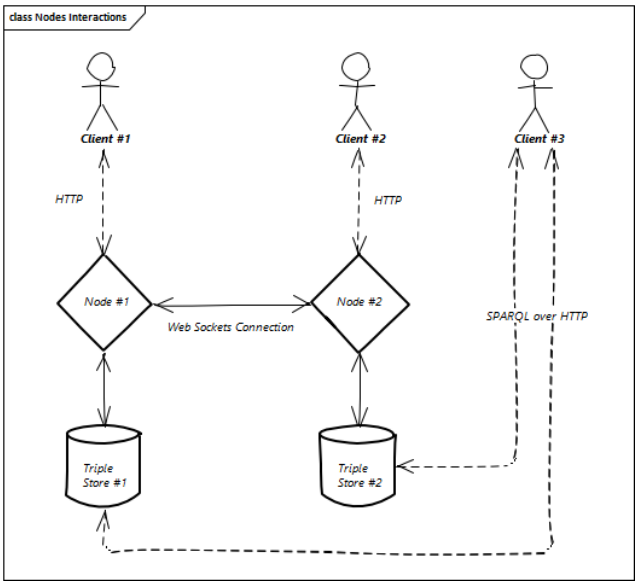

Figure 1: Nodes interaction

so that it would accept other types of serialisation formats. A sample HTTP request is presented in the listing 1. For now we have not decided to limit the size of the input RDF graph, but we will do that in the future works.

```
POST /mammoth/block/create?graphIri=http://lei.info/6SHGI4ZSSLCXXQSBB395
Host: localhost:8881
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:58.0) Gecko/20100101 Firefox/58.0
Content-Type: text/turtle;charset=UTF-8
Content-Length: 36267

@base <graph://lei.info/6SHGI4ZSSLCXXQSBB395>.

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.

<http://lei.info/6SHGI4ZSSLCXXQSBB395> <http://lei.info/gleio/hasInitialRegistrationDate>
        ↪ "2012-06-06T15:53:25.893Z"^^xsd:dateTime;
<http://lei.info/gleio/hasLastManifestation> <http://lei.info/6SHGI4ZSSLCXXQSBB395#
        ↪ GLEIManifestation-20170712>;
<http://lei.info/gleio/hasManifestation> <http://lei.info/6SHGI4ZSSLCXXQSBB395#
        ↪ GLEIManifestation-20151009>,
...
```

Listing 1: Fragment of sample HTTP request with serialised RDF graph. Most of the graph was omitted due to the space limitation of the paper.

After the application receives the graph, it is deserialised and its hash is calculated with one of the methods we will discuss in section 4.3. Then a hash of a string being the concatenation of the following elements: a new block index, a previous block IRI, a previous hash, a current time stamp, a received graph IRI, and the aforementioned graph hash, is computed. The hash is considered to be the hash of the block.

After the hash for the new block is calculated, a new block object is created. It consists of the counterpart Blockchain's header and content (see section 5). The block is then serialised as two sets of triples: one for the header and one for the content. The first set of triples is stored in a *graph ledger* (which is a named graph in a repository) and the second one as a separate named graph. Finally, information about the change is broadcast to the other nodes.

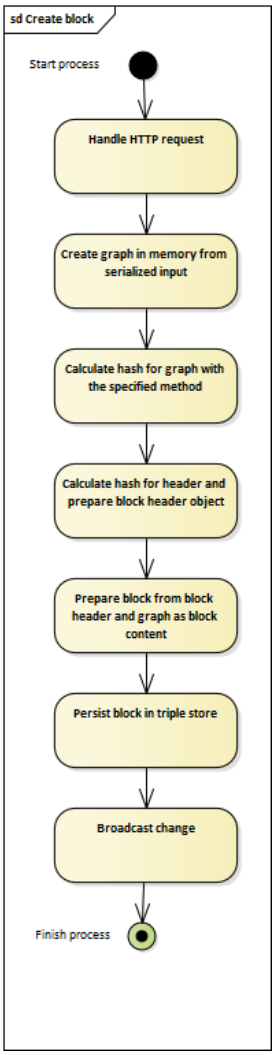

Figure 2: Block creation

4 THE GRAPHCHAIN IMPLEMENTATION CHALLENGES

The implementation of GraphChain as defined and presented in this paper brings a number of challenges that should be addressed before a fully functional, production-grade alternative to the existing Blockchain implementation is offered. These challenges are: 1) Performance of the programmatic access to RDF graphs; 2) Performance and quality of the RDF graphs serialisation used for broadcasting the named graphs to other nodes; 3) Computation of the RDF digests.

4.1 Performance of the programmatic access to the RDF Graphs

While client access to the graphs forming the GraphChain can be realized efficiently using the existing access methods (in most cases using SPARQL), this method may be inefficient for the operations required by the Distributed Ledger Applications. Efficient serialisation mechanisms – important for the broadcasting mechanisms, and fast computation of digests (both issues are addressed below) require as the first pre-requisite, an extremely fast access to "atomic" elements of the graphs, i.e. to triples. While we are not yet presenting in this paper a detailed analysis and comparison between different access methods[9], we assume that in the production grade realisation, the memory-based representations of the RDF graphs shall be used allowing for direct addressing of the individual triples without the overhead of the SPARQL query engines. The detailed analysis of this aspect of the Graphchain implementation will be the subject of our near future studies.

4.2 Performance of RDF graphs serialisations

An efficient broadcasting mechanism needed for the functioning of the Graphchain requires the RDF graphs to be efficiently serialised, broadcasted to the other nodes and then de-serialised to form the internal representation of the graph at all network nodes. We are fully convinced that the best method to achieve high performance serialisation is offered by the HDT serialisation method[10]. However, in the current initial implementations of the GraphChain presented in this paper, we used JSON-LD and Turtle serialisations. It will be replaced by the HDT method in the next steps of the project.

4.3 Computation of RDF digests

Computation of the RDF digest is absolutely essential to the implementation of distributed ledger technology on top of the RDF Graphs. The RDF data model is dramatically different from the block data models used by standard Blockchain implementations and what is needed is cryptographically safe and repeatable computation of the RDF digests.

In [9] the following issues important for computation of the RDF digests have been listed:

(1) Blank nodes' identifiers are implementation dependent, i.e. they may change while transferring the same graph between different implementations, triple stores or other methods of their instantiation.
(2) Every RDF graph is equivalent to an unordered set of triples; so the one and the same graph, even in the same syntax, can be serialised in many different ways.
(3) RDF graph serialisation can be differently encoded; hashing functions are encoding sensitive.

Many attempts have been undertaken in order to find a complete and correct RDF graph hashing algorithm. The history of the attempts is described e.g. in [9, section 2] and recently in [11, section 7] where the following works are mentioned: [1, 4, 5, 8–10, 12, 13, 19, 23].

[9]There is an extensive literature on the efficiency of the triple stores themselves.
[10]See: http://www.rdfhdt.org/.

For this work we have considered three algorithms suitable for the computation of the graph digest:

(1) "Canonicalization" — calculation of the digest as the hash of the canonical graph serialisation
(2) "DotHash" — calculation of the digest as the result of the combining operation on the hashes of the individual triples
(3) "Interwoven DotHash" — calculation of the digest as the result of the combining operation on the hashes of the individual triples and the triples linked by blank nodes.

These algorithms differ in the definition of the block of data submitted to the cryptographic hash function. As the hashing function we used SHA-256 in all of the algorithms.

4.3.1 Canonicalization. This approach represents the most straightforward way in which the digest of the RDF graph can be computed. It is based on the normalised RDF serialisation proposed as the Draft Report of "JSON for Linking Data" W3C Community Group in October 2017[11]. The normalisation algorithm produces a representation of the RDF dataset (collection of the named RDF graphs) with well defined order and with blank nodes represented by canonical identifiers. The normalisation process is designed in such a way, that it should return exactly the same textual representation for the equivalent RDF graphs. In the exemplary implementations described in this paper we used JSON-LD for the normalised graph.

4.3.2 DotHash. The approach to the calculations of RDF Graph digest, named for the purpose of this paper "DotHash" was first presented in [18]. The approach defines the digest of the graph as the result of the combining operation on the hashes of all triples of the graph:

$$\mathcal{D}(S) = \bigodot_{i=1}^{N} h(serialisation(t_i))$$

The combining operation is associative and commutative and allows for the implementation of "incremental cryptography", where the computation of the digest of the graph created by addition of new triples can be done by combining the digest of original graph and the digest of the added triples. Following the analysis by the authors of the approach, the optimal (offering a good compromise between speed and security) approach is based on the "AdHash" algorithm [3] and defines the combining operation as a modulo operation with a suitably large value of the divisor.

The incrementality of the calculations is extremely attractive for the GraphChain as it allows for very efficient implementation in situations permitting new triples to be added to existing graphs in the chain with very low computational overhead. In addition to that, the method allows for highly efficient parallelization even on commodity hardware. However, the method requires a very specific and non-generic approach when the graph contains blank nodes. The authors proposed a method where the blank nodes are labelled on the source side using statements like:

{ _b hasLabel L.}

And these labels are then used on the destination side to rename the blank nodes so that they are identical as on the source side. Of course, then, the calculation of digests on both side will result in the

[11]See https://json-ld.github.io/normalization/spec/

same digests for both graphs. While this approach is practical, we did not find it generic enough and proposed the modified DotHash approach.

4.3.3 Interwoven DotHash. The principle feature of the "Interwoven DotHash" method is its ability to compute the graph digest without canonicalization of the entire graph nor the use of additional triples with labels.

The method we propose ignores the actual format of the blank nodes for the computation of digests while securing the essential effect of the blank nodes: their ability to weave multiple triples together.

As before, we assume that the digest of the entire graph is computed as the combining operation that is associative, commutative and supporting incremental computations of the digest over the graph. Similarly, in the most optimal realisation, the combining operation can be implemented as a modulo (with a sufficiently large divisor) of the hashes of the triples. The hash of the triple can be computed over the entire serialised triple, or as a combination of hashes of the "atomic" nodes of the triple, i.e. the subject, the predicate and the object. This later method allows, in principle, for the use of the alternative fast hash computations for the subject and predicate nodes, as they represent a specific sets of strings from a limited space of values (characteristic for URIs).

Because the blank nodes are anonymous, their actual form and names do not matter. What does matter is the weaving of the triples together into specific constructs used in RDF files (lists, sequences etc).

Exploring these features further we propose the following algorithm for the triple computation:

(1) If the triple does not contain a blank node, compute it as a hash of its N3 serialised format.
(2) If the triple contains a blank node as its Subject, then compute its hash as the sum of the hashes of the N3 serialised Predicate and Object and the hashes of non-blank nodes of all those triples where the blank node appears in the Object nodes.
(3) If the triple contains a blank node as its Object, then compute its hash as the sum of the hashes of the N3 serialised Subject and Predicate and the hashes of non-blank nodes of all those triples where the blank node appears in the Subject nodes.
(4) If the triple contains blank nodes in both Subject & Object nodes, use the above rules twice, once for Subject, then for Objects.

Our algorithm has some similarity to the one proposed in [10, 11]. The difference is in the way we compute the hashes on the level of the triple.

The pseudo-code of the algorithm reads:

```
function Hash(triple)
  subject = subject of triple
  predicate = predicate of triple
  object = object of triple

  if subject is BNode then
    serialisation(subject) = "Magic_S"
  else
    serialisation(subject) = NTriples(subject)

  if object is BNode then
    serialisation(object) = "Magic_O"
  else
    serialisation(object) = NTriples(object)

  serialisation(predicate) = NTriples(predicate)
  concatenation = Concatenate(serialisation(subject), serialisation(predicate), serialisation(object))
  return SHA−256(concatenation)

for triple in graph do
  basic_triple_hash = Hash(triple)
  subject1 = subject of triple
  predicate1 = predicate of triple
  object1 = object of triple

  if subject1 is BNode then
    for all triples with subject1 in object position
      compute Hash(triple)
      add to total_hash

  if object1 is BNode then
    for all triples with object1 in subject position
      compute Hash(triple)
      add to total_hash
```

Listing 2: Interwoven DotHash

5 GRAPHCHAIN ONTOLOGY

The GraphChain ontology[12] is an (OWL) ontology of chained named RDF graphs resembling a sequentially ordered and cryptographically secured chain structure presented in the original paper [15] by Nakamoto.

From the ontological perspective GraphChain's block (unit) is a reified 7-ary relation (see [16, Use Case 3]). In the reification pattern each n-ary relation between resources is represented as a separate OWL class and each instance of the relation—an n-tuple—is represented as an instance of the class plus n additional binary relations providing links to each argument of the n-tuple. The GraphChain's unit is composed of the elements that are listed in the lines 9-15 and 18-24 of listing 3. Lines 8-15 and 17-24 present two instances of blocks `gc:b0` and `gc:b1` and seven triples the blocks are the subjects of. As one can observe GraphChain's unit has counterparts of the elements that normally constitute the Blockchain's block header. In our representation, for simplicity's sake, we have not made an explicit distinction between a block and its header.

The GraphChain units are intended to "store" RDF triples. Strictly speaking, every block has a link to a named graph IRI and the hash of the graph.

The genesis unit is hardcoded and contains the identifier of the GraphChain ontology and its hash. It means that the genesis unit of GraphChain contains information about its semantic structure.

```
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22−rdf−syntax−ns#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf−schema#> .
@prefix gc: <http://www.ontologies.makolab.com/bc/> .

gc:b0 a gc:GenesisBlock ;
  gc:hasDataGraphIRI  "http://www.ontologies.makolab.com/bc"^^xsd:anyURI ;
  gc:hasDataHash "7dc25665a24a48dca4c89e9ba0fe053009d1dc85eaf1fd5c8f5126aa13e3c217" ;
  gc:hasHash "27f9ac0be5bd0fb1a84e74247cb6e5cbe9d49d1692e37a481d4710617cf871c6" ;
  gc:hasIndex "0"^^xsd:decimal ;
  gc:hasPreviousBlock gc:b0 ;
  gc:hasPreviousHash "0" ;
  gc:hasTimeStamp "1502269780"^^xsd:decimal .

gc:b1 a gc:Block ;
```

[12] http://ontologies.makolab.com/bc

```
gc:hasDataGraphIRI "http://makolab.com/foo"^^xsd:anyURI ;
gc:hasDataHash "ea8ea1a9dade4880445bea3e7efe505276a3a0cf14b0fcddf4b5e105012d0edf" ;
gc:hasHash "d50f6fa69a7ff6e1b6cb20ecf87dfff360190c1f9b4fc7ad7f3724bc17f85664" ;
gc:hasIndex "1"^^xsd:decimal ;
gc:hasPreviousBlock gc:b0;
gc:hasPreviousHash "27f9ac0be5bd0fb1a84e74247cb6e5cbe9d49d1692e37a481d4710617cf871c6" ;
gc:hasTimeStamp "1515745336"^^xsd:decimal .
```

Listing 3: The genesis and the first block example

GraphChain consists of 2 classes, 1 object property and 6 data properties. It has ALQ(D) expressivity and 11 restrictions on classes.

6 EXEMPLARY IMPLEMENTATIONS

We developed three implementations presenting the main features of the GraphChain idea.

The C# implementation uses .NET Core platform for ease of deployment in multiple environments. The node itself is an ASP.NET Core web application with REST API for Client - Node communication and WebSocket layer for Peer to Peer transmission. It uses the DotHash algorithm for graph digests. Graph data itself is transferred through websockets in string form as Turtle serialisation encoded in Base64. Local data repository can be either in-memory-store or a dedicated triplestore (e.g. AllegroGraph). The library used as middleware for graph and triples operations is DotNetRdf.

The Java implementation was created as a Spring-managed web application. It uses the RDF4J library for handling semantic-related operations. Currently, this implementation can store RDF graphs in both the RDF4J triple store and the AllegroGraph triple store, but adding a new storage method is possible. Similarly to the C# implementation, the Java implementation uses the DotHash algorithm for graph digests, but it can also use the Canonicalization method for that purpose.

We have also developed a third, illustrative implementation of the node: a JavaScript implementation which is based on Naivechain[13]. It offers HTTP API (https://github.com/lhartikk/naivechain#http-api) and P2P communication between nodes (https://github.com/lhartikk/naivechain#key-concepts-of-naivechain). There are a few differences between our implementation and Naivechain. Our implementation: 1) has different structure of the block: it contains RDF URI referring to graph data and hash of graph data; 2) stores data in dedicated triple store (e.g RDF4J) instead of memory store; 3) includes transferring graph data (referred by IRI) in the process of replication between nodes.

In all reported cases, we used LEI reference data represented as named RDF graphs to test the operations of the Graphchain. We were able, on all participating nodes, to use SPARQL queries on the entire chain of named graphs resulting in the same query results. Depending on the storage mechanism, we were also able to access the graphs with respective native access methods (e.g. iterating over graphs for in-memory storage models).

In all these initial test implementations we intentionally set aside other important features of Blockchain, like the chain update strategy (e.g. LCR - the Longest Chain Rule) or consensus mechanism (e.g. Proof-of-Work or Proof-of-Authority). The motivation for that omission comes from the fact that the implementation of these features does not depend on the actual data model of the chain, because they form another layer of the Blockchain software stack. So implementing that in the initial work would complicate the implementation without benefits for the understanding of the role of the new graph-based data model for the technology.

We have also created a very simple web interface for browsing the content of GraphChain using simple RESTful interface used in the aforementioned implementations[14]. The web interface can be seen in figure 3. It also allows for retrieving data from a graph ledger by means of SPARQL queries. Each node has its own SPARQL Endpoint. Below there are two exemplary queries[15]:

```
PREFIX : <http://www.ontologies.makolab.com/bc/GraphChain/>
PREFIX bc: <http://www.ontologies.makolab.com/bc/>
SELECT * WHERE {
 GRAPH bc:GraphChainJava {
   <http://www.ontologies.makolab.com/bc/GraphChainJava/6> bc:hasDataGraphIRI ?
       ↪ dataGraphAnyURI.
   BIND(IRI(?dataGraphAnyURI) as ?dataGraphIRI) .
   GRAPH ?dataGraphIRI { ?sub ?pred ?obj }
 }
}
```

Listing 4: Select the content of the block number 6. It will be an LEI graph.

```
PREFIX : <http://www.ontologies.makolab.com/bc/GraphChain/>
PREFIX bc: <http://www.ontologies.makolab.com/bc/>
SELECT * WHERE {
 GRAPH bc:GraphChainJava {  <http://www.ontologies.makolab.com/bc/GraphChainJava/6> ?
       ↪ pred ?obj }
}
```

Listing 5: Select "metadata" of the block number 6.

7 RELATED WORKS

Many interesting ideas concerning how blockchain and Semantic Web technologies can be successively combined are discussed in [7]. In this section we shall focus on the main attempts that are aimed at creating a semantic Blockchain.

BLONDiE is an ontology developed by Héctor Ugarte in the Semantic Blockchain project [24]. The OWL ontology is available in GitHub repository: https://github.com/hedugaro/Blondie. It consists of 21 classes, 11 object properties, 50 data properties and 1 annotation property. It has ALCRIF(D) expressivity. We share the opinion expressed in [22] that "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." But BLONDiE as an ontology leaves much to be desired. It specifies the meaning of its classes and properties poorly. Except for disjointedness axioms it contains no other restrictions on classes (here we mean necessary and sufficient conditions). Object properties are characterised as functional, inverse functional, transitive, symmetric, asymmetric reflexive or irreflexive. Each object and data property has its domain and range. BLONDiE is missing some of the properties that are important and specific for our blockchain implementation, e.g. BLONDiE provides no references to the data graph IRI and the previous block IRI.

[13] Naivechain's GitHub: https://github.com/lhartikk/naivechain.

[14] http://binsem.makolab.pl/gcgui/

[15] The queries can be used to retrieve data from the node http://binsem.makolab.pl/gcgui/8881 by:

- http://ml.ms/GraphChain$_{Q}$uery1
- http://ml.ms/GraphChain$_{Q}$uery2

Agent name	Blockchain				
	index	timestamp	dataGraphIri	previousHash	hash
binsem:8881/mammoth/	7	1519723889	http://lei.info/6SHGI4ZSSLCXXQSBB395	f5e9811aacf5575df4997ae0e1d58aa1acb2d64afbada4f8923f341d5ae4d7a3	b8652eee7af392eb33d83230ba07b6c04a6f1c172392249acd703eebbd34a467
	6	1517398832	http://lei.info/PBLD0EJDB5FWOLXP3B76	0f3af827baf5c02049ef5f58f92b94607ddf2c9ddb50b587c44c5ddebf228ba1	f5e9811aacf5575df4997ae0e1d58aa1acb2d64afbada4f8923f341d5ae4d7a3
	5	1517395205	http://lei.info/8I5DZWZKVSZI1NUHU748	86acdb0d782134fceb63cfcc15c9e7a0802bdc6ab3bed46d064c6540dc8f91e4	0f3af827baf5c02049ef5f58f92b94607ddf2c9ddb50b587c44c5ddebf228ba1
	4	1517395019	http://lei.info/F5WCUMTUM4RKZ1MAIE39	117232d272f0eacef9e31442257e3319fde98e0f77e0c14a86206fcf22b059af	86acdb0d782134fceb63cfcc15c9e7a0802bdc6ab3bed46d064c6540dc8f91e4
	3	1516614502	http://lei.info/PBLD0EJDB5FWOLXP3B76	0b5c8a81f892dc4c609303204e969dc603fc3e1a720e45f4119bf22776dd7983	117232d272f0eacef9e31442257e3319fde98e0f77e0c14a86206fcf22b059af
	2	1516613042	http://lei.info/MLU0ZO3ML4LN2LL2TL39	3ecc0180a4d7a4a1a7af94f8d575a55a68fe45db97568845311310925 82c1f6a	0b5c8a81f892dc4c609303204e969dc603fc3e1a720e45f4119bf22776dd7983
	1	1516379147	http://lei.info/2594007XIACKNMUAW223	f4e5f8d34e03c109ae98f6a57cf36fcd6bcf028e1a8362006c1d3bb1500a9c06	3ecc0180a4d7a4a1a7af94f8d575a55a68fe45db97568845311310925 82c1f6a
	0	1502269780	http://www.ontologies.makolab.com/bc	0	f4e5f8d34e03c109ae98f6a57cf36fcd6bcf028e1a8362006c1d3bb1500a9c06
binsem:8883/mammoth/	7	1519723889	http://lei.info/6SHGI4ZSSLCXXQSBB395	f5e9811aacf5575df4997ae0e1d58aa1acb2d64afbada4f8923f341d5ae4d7a3	b8652eee7af392eb33d83230ba07b6c04a6f1c172392249acd703eebbd34a467
	6	1517398832	http://lei.info/PBLD0EJDB5FWOLXP3B76	0f3af827baf5c02049ef5f58f92b94607ddf2c9ddb50b587c44c5ddebf228ba1	f5e9811aacf5575df4997ae0e1d58aa1acb2d64afbada4f8923f341d5ae4d7a3
	5	1517395205	http://lei.info/8I5DZWZKVSZI1NUHU748	86acdb0d782134fceb63cfcc15c9e7a0802bdc6ab3bed46d064c6540dc8f91e4	0f3af827baf5c02049ef5f58f92b94607ddf2c9ddb50b587c44c5ddebf228ba1
	4	1517395019	http://lei.info/F5WCUMTUM4RKZ1MAIE39	117232d272f0eacef9e31442257e3319fde98e0f77e0c14a86206fcf22b059af	86acdb0d782134fceb63cfcc15c9e7a0802bdc6ab3bed46d064c6540dc8f91e4
	3	1516614502	http://lei.info/PBLD0EJDB5FWOLXP3B76	0b5c8a81f892dc4c609303204e969dc603fc3e1a720e45f4119bf22776dd7983	117232d272f0eacef9e31442257e3319fde98e0f77e0c14a86206fcf22b059af
	2	1516613042	http://lei.info/MLU0ZO3ML4LN2LL2TL39	3ecc0180a4d7a4a1a7af94f8d575a55a68fe45db97568845311310925 82c1f6a	0b5c8a81f892dc4c609303204e969dc603fc3e1a720e45f4119bf22776dd7983
	1	1516379147	http://lei.info/2594007XIACKNMUAW223	f4e5f8d34e03c109ae98f6a57cf36fcd6bcf028e1a8362006c1d3bb1500a9c06	3ecc0180a4d7a4a1a7af94f8d575a55a68fe45db97568845311310925 82c1f6a
	0	1502269780	http://www.ontologies.makolab.com/bc	0	f4e5f8d34e03c109ae98f6a57cf36fcd6bcf028e1a8362006c1d3bb1500a9c06

Figure 3: Web interface for browsing the content of GraphChain's graph ledger

EthOn is an ontology that is intended to be a semantical counterpart of the Ethereum Blockchain framework. It contains 46 classes, 48 object, 60 datatype properties and 3 annotation properties. Its documentation is here: http://ethon.consensys.net and other resources are stored in GitHub: https://github.com/ConsenSys/EthOn. EthOn supports multiple inheritance (e.g. `Block` is a child class of `StateTransition` and `BlockConcept`). EthOn has a few disjointedness axioms and 9 restrictions — 2 for `Account` class and 7 for subclasses of `Msg` class (involving 3 object properties: `from`, `to` and `creates`). Its potential has not been explored enough yet. For instance, more restrictions could be added to contribute more to the meaning of the EthOn classes, for instance, it could be made explicit (what is currently written only in the comments) that each `AccountStorage` is part of some `AccountState`, etc. EthOn taxonomy is not correct. It is common that the authors mix metalevel classes with the domain ones, e.g. `Block` is a subclass of `BlockConcept` meaning that each block is a block concept, and that is simply not true.

Flex (Web) Ledger is a graph data model and a protocol for decentralized ledgers [14, 20, 21]. From the data model perspective, Flex Ledger assumes the use of generic JSON objects encapsulated in the ledger. But "The ledger data model and syntax make no assumption about which ontology is used." [14, p.1441]. So Flex Ledger does not have its own ontology and each concrete implementation of the ledger requires additional specification on the level of data structure and meaning. It is also worth mentioning that the Flex Ledger meta and the content data are stored together in the same graph whereas the GraphChain blocks' content is store outside the block in a separate graph. For all the reasons stated above, GraphChain cannot be considered as an implementation of Flex Ledger.

There is another very important related development that parallels GraphChain although from a different perspective than Semantic Web: MongoDB has recently announced the creation of Blockchain infrastructure combined with the existing MongoDB database. The similarity to our approach comes from the fact that Blockchain layer in the proposal does not change the native mechanisms to access data stored on MongoDB database, as is GraphChain not changing the access mechanisms native to RDF databases[16].

8 CONCLUDING REMARKS

In this paper, we have proposed a new approach to the architecture of a Blockchain compliant system, where the fundamental data structure is based on RDF graphs and where semantic technologies, including OWL ontologies and Linked Data principles form the foundations of its architecture. The emerging new Blockchain platform is called GraphChain.

GraphChain is defined on the level of the abstract RDF data model. Such high level definition opens the possibility for its implementation as a software layer independent of the actual implementation of RDF data storage, querying mechanisms or serialisations. It also guarantees that all access methods characteristic for the given data storage remain unchanged, making the data fully available and open. We have demonstrated that the key elements of distributed ledger technology, as are usually implemented in Blockchain, can be implemented in GraphChain. In particular, serialisations required for broadcasting a graph into the network and computation of the graph digest can be implemented without compromising generality of the approach.

The key element of the GraphChain is its ontology. It is in this part of the proposal where we define, also on the high level of abstraction, how the specific elements of Blockchain technology are to be implemented in the GraphChain Ledger. This ontological aspect of the emerging platform, combined with ontologies for data stored in the chain of named graphs, create a system that is fully compliant with semantic technologies and exhibits all the benefits of Distributed Ledger Technologies as implemented in a standard Blockchain.

We are also reporting here three simple implementations of the GraphChain using Java, .NET and node.js technologies. The

[16] https://www.mongodb.com/collateral/building-enterprise-grade-blockchain-databases-with-mongodb

implementations were, by design, extremely simple and were only used for the verification of GraphChain's foundational features.

The work is now in progress to use qualified open source Blockchain platform and create a production grade GraphChain implementation. Looking toward the production grade GraphChain, we are also exploring the possibilities for high-speed transactions on GraphChain. This is possible due to high level of parallelisation possible for the key elements of the future solutions.

The production grade editions of the GraphChain will first be used for the creation of distributed ledger editions of the digital identity applications for Legal Entity Identifier (LEI) system related to the LEI.INFO initiative (http://lei.info), in its strife to create radically new foundations of the global LEI system.

REFERENCES

[1] Jesús Arias-Fisteus, Norberto Fernández García, Luis Sánchez Fernández, and Carlos Delgado Kloos. 2010. Hashing and canonicalizing Notation 3 graphs. *J. Comput. Syst. Sci.* 76, 7 (2010), 663–685. https://doi.org/10.1016/j.jcss.2010.01.003

[2] Rick Barrett, Rick Cummings, Eugene Agichtein, and Evgeniy Gabrilovich (Eds.). 2017. *Proceedings of the 26th International Conference on World Wide Web Companion, Perth, Australia, April 3-7, 2017.* ACM. http://dl.acm.org/citation.cfm?id=3041021

[3] Mihir Bellare and Daniele Micciancio. 1997. A New Paradigm for Collision-Free Hashing: Incrementality at Reduced Cost. In *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding (Lecture Notes in Computer Science)*, Walter Fumy (Ed.), Vol. 1233. Springer, 163–192. https://doi.org/10.1007/3-540-69053-$0_1$3

[4] Jeremy J. Carroll. 2003. *Signing RDF Graphs.* Technical Report HPL-2003-142. Hewlett-Packard Labs.

[5] Jeremy J. Carroll, Christian Bizer, Pat Hayes, and Patrick Stickler. 2005. Named graphs, provenance and trust. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*. ACM, New York, NY, USA, 613–622. https://doi.org/10.1145/1060745.1060835

[6] Nell Dale and Henry M. Walker. 1996. *Abstract data types: specifications, implementations, and applications.* Jones & Bartlett Learning.

[7] M. English, S. Auer, and S.Domingue. 2016. Blockchain technologies & the Semantic Web: A framework for symbiotic development. In *CS Conference for University of Bonn Students*, J. Lehmann, H. Thakkar, L. Halilaj, and R. Asmat (Eds.). 47–61.

[8] Mark Giereth. 2005. On Partial Encryption of RDF-Graphs. In *The Semantic Web - ISWC 2005, 4th International Semantic Web Conference, ISWC 2005, Galway, Ireland, November 6-10, 2005, Proceedings (Lecture Notes in Computer Science)*, Yolanda Gil, Enrico Motta, V. Richard Benjamins, and Mark A. Musen (Eds.), Vol. 3729. Springer, 308–322. https://doi.org/10.1007/11574620$_2$4

[9] Edzard Höfig and Ina Schieferdecker. 2014. Hashing of RDF Graphs and a Solution to the Blank Node Problem. In *Proceedings of the 10th International Workshop on Uncertainty Reasoning for the Semantic Web (URSW 2014) co-located with the 13th International Semantic Web Conference (ISWC 2014), Riva del Garda, Italy, October 19, 2014. (CEUR Workshop Proceedings)*, Fernando Bobillo, Rommel N. Carvalho, Davide Ceolin, Paulo Cesar G. da Costa, Claudia d'Amato, Nicola Fanizzi, Kathryn B. Laskey, Kenneth J. Laskey, Thomas Lukasiewicz, Trevor P. Martin, Matthias Nickles, Michael Pool, Tom De Nies, Olaf Hartig, Paul T. Groth, and Stephen Marsh (Eds.), Vol. 1259. CEUR-WS.org, 55–66. http://ceur-ws.org/Vol-1259/method2014$_s$ubmission$_1$.pdf

[10] Aidan Hogan. 2015. Skolemising Blank Nodes while Preserving Isomorphism. In *Proceedings of the 24th International Conference on World Wide Web, WWW 2015, Florence, Italy, May 18-22, 2015*, Aldo Gangemi, Stefano Leonardi, and Alessandro Panconesi (Eds.). ACM, 430–440. https://doi.org/10.1145/2736277.2741653

[11] Aidan Hogan. 2017. Canonical Forms for Isomorphic and Equivalent RDF Graphs: Algorithms for Leaning and Labelling Blank Nodes. *ACM Trans. Web* 11, 4, Article 22 (July 2017), 62 pages. https://doi.org/10.1145/3068333

[12] Andreas Kasten, Ansgar Scherp, and Peter Schauß. 2014. A Framework for Iterative Signing of Graph Data on the Web, See [17], 146–160. https://doi.org/10.1007/978-3-319-07443-$6_1$1

[13] Tobias Kuhn and Michel Dumontier. 2014. Trusty URIs: Verifiable, Immutable, and Permanent Digital Artifacts for Linked Data, See [17], 395–410. https://doi.org/10.1007/978-3-319-07443-$6_2$7

[14] Victoria L. Lemieux and Manu Sporny. 2017. Preserving the Archival Bond in Distributed Ledgers: A Data Model and Syntax, See [2], 1437–1443. https://doi.org/10.1145/3041021.3053896

[15] Satoshi Nakamoto. 2008. *Bitcoin: A Peer-to-Peer Electronic Cash System.* Technical Report. http://www.bitcoin.org/bitcoin.pdf

[16] Natasha Noy and Alan Rector. 2006. *Defining N-ary Relations on the Semantic Web.* W3C Working Group Note. World Wide Web Consortium. http://www.w3.org/TR/swbp-n-aryRelations/

[17] Valentina Presutti, Claudia d'Amato, Fabien Gandon, Mathieu d'Aquin, Steffen Staab, and Anna Tordai (Eds.). 2014. *The Semantic Web: Trends and Challenges - 11th International Conference, ESWC 2014, Anissaras, Crete, Greece, May 25-29, 2014. Proceedings.* Lecture Notes in Computer Science, Vol. 8465. Springer. https://doi.org/10.1007/978-3-319-07443-6

[18] C. Sayers and A. H. Karp. 2003. *Computing the Digest of an RDF Graph.* Technical Report HPL-2003-235. Hewlett-Packard Labs.

[19] C. Sayers and A. H. Karp. 2004. *RDF Graph Digest Techniques and Potential Applications.* Technical Report HPL-2004-95. Hewlett-Packard Labs.

[20] Manu Sporny and Dave Longley. 2016. *Flex Ledger 1.0: A flexible format and protocol for decentralized ledgers on the Web.* Technical Report. Digital Bazaar, https://digitalbazaar.github.io/flex-ledger/.

[21] Manu Sporny and Dave Longley. 2017. *The Web Ledger Protocol 1.0 A format and protocol for decentralized ledgers on the Web.* Technical Report. Digital Bazaar, https://w3c.github.io/web-ledger/.

[22] Allan Third and John Domingue. 2017. Linked Data Indexing of Distributed Ledgers, See [2], 1431–1436. http://dl.acm.org/citation.cfm?id=3041021

[23] Giovanni Tummarello, Christian Morbidoni, Paolo Puliti, and Francesco Piazza. 2005. Signing individual fragments of an RDF graph. In *Proceedings of the 14th international conference on World Wide Web, WWW 2005, Chiba, Japan, May 10-14, 2005 - Special interest tracks and posters*, Allan Ellis and Tatsuya Hagino (Eds.). ACM, 1020–1021. https://doi.org/10.1145/1062745.1062848

[24] Hector Ugarte. 2017. *A more pragmatic Web 3.0: Linked Blockchain Data.* Technical Report. https://www.researchgate.net/publication/315619465_A_more_pragmatic_Web_30_Linked_Blockchain_Data.

[25] Roger Wattenhofer. 2016. *The Science of the Blockchain* (1st ed.). CreateSpace Independent Publishing Platform, USA.