Standardized and Efficient RDF Encoding for Constrained Embedded Networks

Sebastian Käbisch(✉), Daniel Peintner, and Darko Anicic

Siemens AG, Corporate Technology, Munich, Germany
{Sebastian.Kaebisch,Daniel.Peintner.ext,Darko.Anicic}@siemens.com

Abstract. In the context of Web of Things (WoT), embedded networks have to face the challenge of getting ever more complex. The complexity arises as the number of interchanging heterogeneous devices and different hardware resource classes always increase. When it comes to the development and the use of embedded networks in the WoT domain, Semantic Web technologies are seen as one way to tackle this complexity. For example, properties and capabilities of embedded devices may be semantically described in order to enable an effective search over different classes of devices, semantic data integration may be deployed to integrate data produced by these devices, or embedded devices may be empowered to reason about semantic data in the context of WoT applications. Despite these possibilities, a wide adoption of Semantic Web or Linked Data technologies in the domain of embedded networks has not been established yet. One reason for this is an inefficient representation of semantic data. Serialisation formats of RDF data, such as for instance a plain-text XML, are not suitable for embedded devices. In this paper, we present an approach that enables constrained devices, such as microcontrollers with very limited hardware resources, to store and process semantic data. Our approach is based on the W3C Efficient XML Interchange (EXI) format. To show the applicability of the approach, we provide an EXI-based μRDF Store and show associated evaluation results.

Keywords: Web of Things (WoT) · Microcontroller · RDF · EXI · RDF store

1 Introduction

We are witnessing a new era of innovation which is taking place through the convergence of the physical and cyber world. This era is characterised with an emergence of technologies such as low-cost sensing, smart devices, advanced computing, powerful analytics, and the new levels of connectivity permitted by the Internet - all together often referred to as Internet of Things (IoT). Further integration of physical devices and the data they produce with the Web is also known as the Web of Things (WoT).

While it has a huge potential to change our lives in various aspects, WoT still faces a number of challenges such as for example identification and discovery of

© Springer International Publishing Switzerland 2015
F. Gandon et al. (Eds.): ESWC 2015, LNCS 9088, pp. 437–452, 2015.
DOI: 10.1007/978-3-319-18818-8_27

WoT devices and services, machine interpretation and integration of WoT data, automated interactions of WoT devices in a certain context and others.

Semantic Web (SW) technologies are seen as a good candidate to tackle these and other challenges in the realm of WoT. The W3C Resource Description Framework (RDF) [19] is a powerful data model that is used for conceptual descriptions and modelling of information in the Web. RDF expressions, provided in a form of subject-property-object triples, represent statements about (Web) resources. In the context of WoT, resources are physical 'things' (e.g., sensors, actuators, etc.) that are connected to the Web and can be in the same way described as a set of RDF statements. RDF descriptions, written in accordance to a certain schema or ontology, may later help in discovery of WoT devices with certain characteristics. This can reduce the time of building new applications significantly, as for example the semantic search can be used for this task. The data produced by selected devices may be easier integrated and processed, thereby creating a new information or an added-value service. A WoT device can also be easier integrated into a running system if both the device and the contextual information of the system are semantically described. Thanks to semantic reasoning, that enables the WoT device to find its role in the system, it also enables the device to demonstrate a plug&play functionality in an WoT environment.

Despite these few examples, a straight forward use of SW technologies in the context of Web of Things applications is not possible. Typical devices, associated with physical 'things', are very limited in terms of their capabilities (i.e., processing power, available memory, energy supply etc.). For example, WoT devices, such as sensors and actuators run by microcontrollers with only few kilo Bytes of RAM and ROM and have a slow processing unit (e.g., ARM Cortex-M3 microcontroller[1]), are not capable to store and process RDF triples serialized in formats such as plain-text RDF/XML.

In the recent of years there has been many efforts to find a format to compress huge sets of RDF triples (e.g., HDT [7] and RDSZ [8]). Although the compression results are respectful in terms of the decrease of the network traffic, these approaches however do not target very constrained devices such as microcontrollers. Two reasons hinder them in this goal, and those are memory usage and processing constrains, imposed by tiny devices such as for example ARM Cortex-M3 microcontroller.

Consequently, requirements related to an RDF serialization format for constrained devices, and at the same time, for the use of SW technologies in the realm of WoT, should fulfil the following aspects:

- **Low Memory Usage**: the memory used for semantic descriptions should be as small as possible and should always leave enough space for the actually run time procedure.
- **Small Message Size**: in embedded networks the bandwidth usage can be very critical, hence transferred messages should be kept small.
- **Type Awareness**: physical devices will mainly exchange physical values with certain characteristics (unit of measure, precision, sampling rate etc.), hence

[1] http://www.arm.com/products/processors/cortex-m/cortex-m3.php.

the data shall be represented in a type aware manner. Type awareness optimizes the overall processing and reduces the memory usage.
- **Simple Processing**: small embedded devices shall be enabled to read the content of semantic data in a high efficient and direct manner. For example, the overhead of a transformation of data should be avoided before the actual data content can be retrieved.
- **A Standardized Solution**: To avoid or reduce the development effort and costs such as found in proprietary solutions, a standardized approach for the use of semantics from powerful devices up to tiny constrained embedded devices, should be pursued.

This paper addresses all the above mentioned requirements and proposes an approach that relies on the technique of the standardized W3C's Efficient XML Interchange (EXI) format [17]. It makes the serialization of RDF data efficient and applicable, even for very constrained embedded devices.

The paper presents the following contributions and is organised as follows.

- We start to give an overview about related work in Sect. 2 and discuss its intricacy in terms of its applicability in the microcontroller environment.
- In Sect. 3 we introduce the W3C EXI format and our different proposals to serialize RDF-based data in a efficient manner.
- To show the applicability of our approach in the embedded domain, we have developed μRDF Store - a repository that stores and serializes semantic data in the EXI format. This work is detailed in Sect. 4.
- Finally, we present evaluation results showing the effectiveness of our approach to encode RDF with EXI. Further on, we prove its applicability to the constrained embedded domain such as the one with microcontrollers (Sect. 5).

2 Related Work

The related work in this topic can be parted into two main subjects, namely the effort of existing RDF compression approaches and the existing semantic repositories.

2.1 RDF Compressions

HDT and SHDT. Header-Dictionary-Triples (HDT) [7] is a well known binary format for publishing and exchanging of RDF data. The main idea behind the approach is to decompose an RDF document into a Header-Dictionary-Triples (HDT) format, and represent it in a compact manner, thereby decreasing the redundancy in an RDF graph. The HDT format consists of: a *Header*, a *Dictionary*, and *Triples*. The Header includes optional metadata that describes the RDF dataset. The Dictionary provides a vocabulary of the RDF terms, i.e., a catalogue where for each distinct term, a unique ID is assigned. This way, the dictionary contributes to the goal of compactness by replacing the long repeated

strings in triples by short IDs. IDs can be used for indexing of RDF data too. The triples component comprises the pure structure of the underlying RDF graph, i.e., compactly encodes the set of triples while avoiding the noise produced by long labels and repetitions. In this way an original RDF triple can be expressed as three IDs, thereby replacing each element in a triple with the reference to the dictionary. An experimental evaluation of a concrete implementation from [7] shows that datasets in the HDT format can be compacted by more than fifteen times as compared to a naive representation. Specific compression techniques over HDT, such as for example Huffman [12] and PPM [3] encoding, may further improve these compression rates. However, this is also implemented at the expense of additional processing overhead which is not feasible to constrained embedded devices with very limited memory and processing capability (e.g., microcontroller ARM Cortex-M3).

Streaming HDT (SHDT) [10] further extends the original HDT format toward a format which is better suited for a streaming data. For big documents that cannot fit into memory, SHDT avoids full assembling of the dictionary before it starts writing triples, and does not need to collect all triples of a document to create the graph encoding. Instead, a document is assembled on-the-fly as a stream of chunks with sizes that depend on available memory. As such, the SHDT approach would be also more suitable for an embedded environment, since the implementation complexity and memory usage (no buffer for assembling the dictionary is required any more) is lower than with the native HDT approach. Unfortunately, HDT and SHDT are not compatible to each other. The incompatibility arises from the fact that SHDT can re-use IDs while encoding, and for the HDT format this is not the case.

In general, both approaches are focusing on encoding of RDF data represented as strings, and do not provide potentials for an effective data-type aware encoding. In our view, this is a very important issue. In embedded networks when it comes to a direct machine to machine (M2M) interaction, physical values are mostly typed (e.g., int, boolean, etc.). Using a string based representation of data types (as in RDF) would always lead to an additional processing overhead in type conversion on both sides, encoder's and decoder's side. In addition, there is a missing clarification about the trade-off between the dynamic RAM size of the directory and the message size.

RDSZ. The RDF Differential Stream compressor based on Zlib (RDSZ) [8] approach uses differential encoding to take compressional advantages of the structural similarities in an RDF stream with the general purpose stream compressor Zlib which implements the DEFLATE-algorithm [15]. The major focus is on avoidance of redundancy in the stream. However, this is done at the expense of additional data processing steps, that lead to the lost of the basic RDF triple structure of the produced stream. In addition, the proposed Zlib library is not applicable to small embedded devices such as microcontrollers.

ERI. Efficient RDF Interchange (ERI) [6] is based on RDSZ and an assumption that the structure of the data of RDF streams is predetermined. This structure is determined throughout *Presets* - an information set that identifies, among other things, predicates producing massive data repetitions. The Presets have to be

shared forehand by encoder and decoder in order to take advantage of streamed repeated data. Further on, ERI produces an RDF stream as a continuous flow of blocks of triples where each block is modularized into sets of structural and value channels. For each channel standard compression approaches, such as Zlib, can be applied.

ERI is mainly focused on the compression of the size of RDF data with the goal of decreasing the network traffic. However, an implementation of this approach in the constrained embedded domain would be hard due to the necessary pre-determination of the Presets and their sharing by encoders and decoders, as well as the usage of compression techniques such as Zlib. Furthermore, this approach, similarly as previous approaches, also does not take the advantage of the type aware encoding into account.

RDF Thrift. The open source project RDF Binary using Apache Thrift[2] is a binary format for RDF. The approach defines basic encoding for RDF terms, and then builds formats for RDF graphs, RDF datasets and for SPARQL results. The main goal of RDF Thrift is to enable efficient processing and transfer of RDF data, using Apache Thrift[3] as a non-human-readable data format designed for efficient exchange of data between co-operating processes and interoperable across different programming languages.

2.2 Semantic Repositories

Conventional Stores. There exists a number of RDF repository implementations such as Apache Jena[4], Sesame[5], YARS[6] and many others[7]. For an extensive survey, see also [5]. As mentioned in the introduction section, these implementations have not been suited to run on constrained embedded devices as found in today's IoT/WoT applications.

In the remaining part of this section we give an overview of few RDF repositories that are working with various types of compact representations for RDF.

RDF HDT. An implementation of the earlier described HDT approach is available as an open source project[8]. It is a set of libraries that enable RDF data to be represented, indexed and queried in the HDT format. The project provides inplementations in both C++ and Java, as well as an HDT integration with Apache Jena.

Wiselib TupleStore. An approach which addresses constrained devices is the Wiselib TupleStore [11]. To handle the string-based data of RDF triples, the Huffman compression [12] can be applied. This kind of RDF Store keeps a

[2] http://afs.github.io/rdf-thrift/.
[3] http://thrift.apache.org.
[4] http://jena.apache.org/.
[5] http://rdf4j.org/.
[6] http://sw.deri.org/2004/06/yars/.
[7] http://www.w3.org/wiki/SemanticWebTools#RDF_Triple_Store_Systems.
[8] http://www.rdfhdt.org.

collection of quadruples: subject, predicate, object, and a bit mask which defines to which RDF document the tuple belongs. In terms of serialization, TupleStore supports the SHDT approach as the serialization format (described above).

3 The W3C EXI Format for RDF

Recently the World Wide Web Consortium (W3C), home of XML, was faced with the drawbacks of plain-text XML representation, and hence created a working group called XML Binary Characterization (XBC) [9] to analyze the condition and possibilities of a binary XML format. This format was supposed to be compatible with the standardized plain-text XML format, as well as with the XML Infoset. The outcome was the W3C's Efficient XML Interchange (EXI) format, which gained recommendation status at the beginning of 2011 [17]. The EXI format uses a relatively simple grammar-driven approach that achieves very efficient encodings (EXI streams) for a broad range of use-cases. According to [2] the EXI representation is often over hundred times smaller than the one of XML. Based on the high compression ratio and the opportunity to obtain the typed data content directly from the EXI stream, XML-based messaging is feasible in the embedded domain too, even for very constrained devices [14]. Based on EXI's beneficial characteristics w.r.t. the embedded domain and constrained resources such as memory, processing capability, and bandwidth usage, EXI is getting established more and more in embedded industry applications such as in the domain of automotive industry (e.g., e-Mobility [13]) and smart grid application (e.g., Smart Energy Profile 2 [21]).

As noted above, EXI uses a grammar-driven approach to represent XML-based data in an efficient binary form and vice versa. Such a grammar is constructed according to a given XML Schema where each defined complex type is represented as a deterministic finite automaton (DFA). Moreover, EXI also has the capability to work schema-less, meaning that an EXI processor uses generic grammars provided by the EXI standard.

In the case of RDF/XML representation, it makes sense to use the schema-informed EXI mode given that we know the RDF *schema* and how RDF data looks like [20]: each RDF document starts with the *RDF* root element and nests the set of *Description* child elements to formulate triples. This enables one to formulate various RDF schemas and EXI grammar respectively which can be selected depending on the actual applications. Those kinds of variations will be discussed in the next subsections.

3.1 Generic RDF EXI Grammar

Figure 1 shows an excerpt of a sample EXI grammar (set of automaton) G that can be used for encoding and decoding generic RDF data. This grammar reflects an XML Schema that represents the RDF framework with the *RDF* root element and its embedded *Description* element for representing the triple information. It is worth noting that the *Root* grammar is a predefined grammar

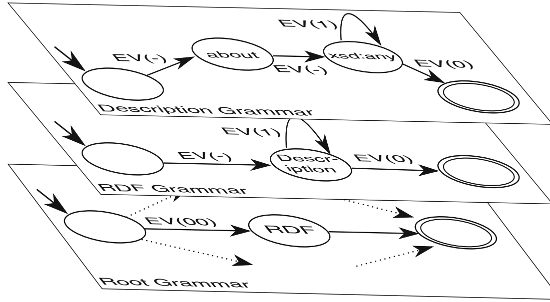

Fig. 1. Generic RDF EXI grammar

that occurs in each EXI grammar representation of arbitrary XML Schemas. It contains all entry points to all root elements in a given schema. Here, we highlight the context of the relevant RDF root element of the RDF XML framework. In general, each DFA contains one start state and one end state, which reflect the beginning and the end, respectively, of a complex type declaration. Transitions to the next state represent the sequential order of element and/or attribute declarations within a complex type. Optional definitions (e.g., *choice*, $minOccurs = '0'$, $maxOccurs = 'unbounded'$ etc.) are reflected by multiple transitions and assigned an event code (EV). For instance, the *Description* element is typically a reoccurring element in a RDF instance and consequently defined as a 'loop' in the XML Schema by $maxOccurs = 'unbounded'$ and reflected by the EXI grammar by the two transitions: one to the *Description* state again and one to the end state. For the signalization, a one bit event code is used and assigned to the transition (EV(1) for the *Description*; EV(0) for no further *Description*). Generally, the number of bits used for m transitions is determined by $\lceil log_2 m \rceil$. EV(-) on transitions indicates, no event code is required.

The *xsd:any* state represents the predicate and object description. This state indicates the generic part since the name of the predicate elements as well as the object values are application dependent.

An example RDF-XML snippet such as the following one:

```
<RDF>
  <Description about= ‘temperature’>
    <type resource=’sensor’/>
    <value>8.4</value>
  </Description>
  <Description about= ’humidity’>
    <type resource=’sensor’/>
    <value>79.2</value>
    ...
```

would be transformed into an:

00 'temperature' type resource 'sensor' value '8.4' 0 1 'humidity' 0 1 0 3 '79.2'...

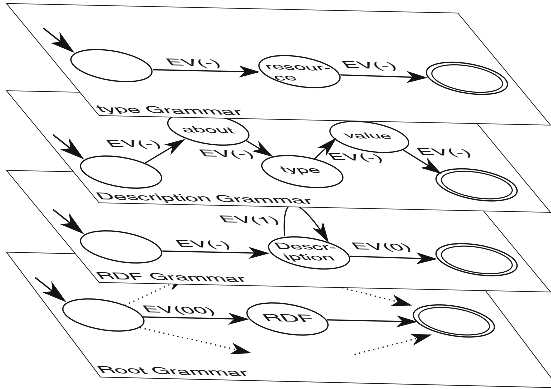

Fig. 2. Application specific RDF EXI grammar

EXI stream[9]. This sketches already how compact an EXI can become compared to the XML counterpart. Blue color indicates the bit-based event codes to navigate the EXI grammar for encoding (and decoding). Green indicates the values of the attributes (e.g., *about* attribute with 'temperature' and 'humidity') and elements (e.g., *value* element with '8.4' and '9'). Generally, EXI is a type-aware coder that provides efficient coding mechanisms for the most common data types (e.g., int, float, enumerations, etc.). For the sake of simplicity, the values are shown in human readable form in the sample EXI stream. Nevertheless, in the case of the float-based value of the temperature context, EXI would only spend two bytes to represent the value '8.4'. The orange content represents the *predicate* elements which are not schema known and covered by the *xsd:any* deceleration. The EXI coding mechanism for that case is based on the following idea: by the first occurrence of an unknown element or attribute the name is provided in the EXI stream (e.g., value). Internally, the string-based name is memorized and an associated unique ID is assigned. That said, for any other appearance of the same string this ID is used instead.

3.2 Application-Specific RDF EXI Grammar

To avoid the usage of such a generic approach that associates, e.g., unknown predicate element names, one can define an XML Schema or EXI grammar respectively. It make sense to follow this approach if the ontology and the context of the semantic description is already known and can be reflected in a schema definition. This can include, e.g., all data and object properties, classes, and possible data value ranges. For example, let us assume there is an ontology for an embedded device that is only intended to serve temperature and humidity values as conducted in the XML snippet above. To represent this semantic requirement, the EXI grammar which is shown in Fig. 2 can be derived. Compared to the grammar shown in Fig. 1 the *Description* automaton is more concrete by the

[9] For the sake of convenience, namespaces will be omitted in the paper.

type and a *value* state between the *about* attribute state and the end state. Applying this to the XML snippet above we would get

`00 'temperature' 0 '8.4' 1 'humidity' 0 '79.2'..`

as EXI stream. It can be immediately seen, this results in a more compact representation since the knowledge about the type, resource, and the value element is already mapped to the grammar, and does not need to be represented in the EXI stream any longer. In addition, since the ontology provides the definition of all classes relevant for the stream, we can define an enumeration list of all class names within the XML Schema. EXI will only use the enumeration value for association, as previously shown for the red 0 (='sensor').

As mentioned before, embedded devices from the industrial automation domain will mainly exchange physical values. String-based values are rare. Therefore we propose to use the W3C EXI Profile for limiting the usage of dynamic memory [4]. This enables us to operate without value string tables. In the case of reoccurring value strings (e.g., 'temperature' and 'humidity'), those strings will be simply handled as *new* strings (given that the possible number of string entries is set to zero).

4 μRDF Store with EXI

In this section we will explain the use of different EXI grammar approaches to efficiently represent and store RDF data in a semantic repository. For this purpose we have implemented a triple store called μRDF Store. To interact and to operate with the μRDF Store, we use a REST-based interface based on the IETF Constrained Application Protocol (CoAP) [18]. Our approach can be compared to RDF Provider of the Wiselib TupleStore [11] (see also Sect. 2.2). In contrast to this, our approach is based on the standardized coding mechanism inherited from the W3C EXI format. The mechanism and the format are used both, to efficiently serialize and to store the data in the RDF repository.

The fundamental concept and the use of the two different grammar variants of our μRDF store are shown in Fig. 3. They both use the same REST-based communication component, implemented with the IETF CoAP approach. So far, we have implemented three CoAP resources to manipulatable data in the μRDF store: *addTriples* to add new or update existing triples, *searchTriples* to enable graph pattern matching, and *deleteTriples* to remove triples from the semantic repository. To serialize the full content of the μRDF store, one can use the *searchTriples* resource with the search pattern (*,*,*). All requested patterns will be answered by an RDF/XML-based document, which is encoded by the EXI format and the underlying EXI grammar, respectively.

The use of the generic grammar and concrete grammar approach in the μRDF store will be explained in the following subsections.

4.1 μRDF Store with Generic Grammar

Apart from the used generic grammar, different string tables can be used to manage unknown element/attributes names (from properties) and string-based values

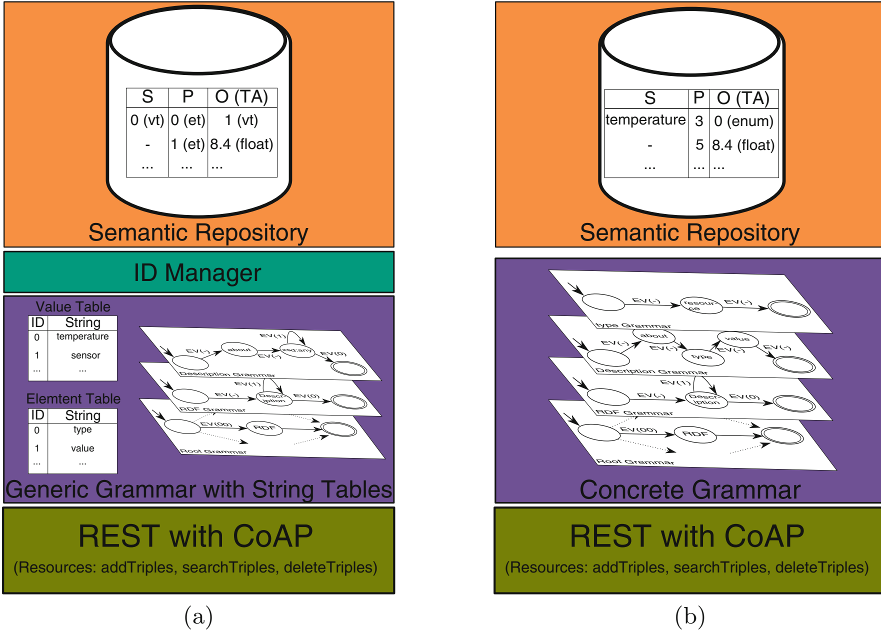

Fig. 3. μRDF using the (a) generic grammar approach with a value table (vt) and an element table (et) and (b) the concrete grammar approach.

(see Sect. 3.1). More precisely, if there is a PUT request on the *addTriples* resource, with a payload of an RDF-based message encoded by EXI (mime-type=EXI), the encoded subject names (provided by the *about* attribute withing the *Description* element) and string-based values of the objects will be affiliated in the value table. Property element names (nested in the RDF Description element) will be managed in the element table. For instance, the EXI snippet below

00 'temperature' type resource 'sensor' ...

identifies the triple with the subject 'temperature' and assigns the ID value 0 throw the value table, predicate 'type' that is assigned by the ID 0 throw the element table, and the object 'sensor' by the ID 1 throw the value table (see also Sect. 3.1). These IDs are used in the semantic repository to represent the triple's subject, property, and object. Based on our running example, the IDs (0,0,1) in the repository corresponds to (temperature, type, sensor).

The *ID Manager* component is mainly used to achieve consistency between the string tables and the repository. This is necessary when triples are deleted from the repository (via the *deleteTriples* resource).

The great advantage of the generic approach is that no requester (e.g., a client) of the μRDF Store service needs to have a pre-knowledge of the content of the semantic repository in order to decode the (result) of an RDF graph. This is possible thanks to the relative generic EXI grammar approach, combined with the EXI's default coding mechanism [17]. The downside of this, is that

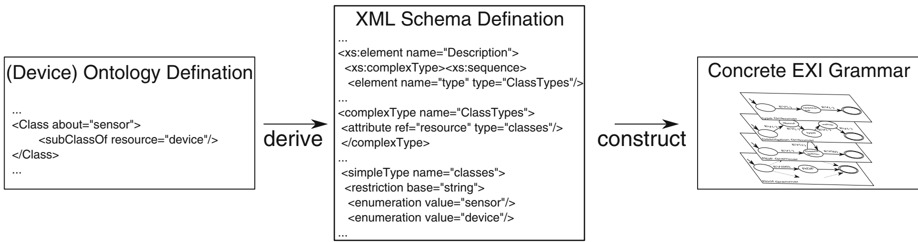

Fig. 4. Grammar construction based on an ontology and XML Schema definition.

more memory has to be available to handle the string based values within the string-tables, as well as to have relatively higher number of string representations within the RDF-based serialized messages (see Sect. 3.1). The next subsection explains how we can overcome this issue by using a concrete EXI grammar.

4.2 μRDF Store with Concrete Grammar

String tables, as used in the generic approach, are not necessary in the concrete grammar approach. This is justified by the fact that almost all triple graph structures are known from a defined ontology (see Sect. 3.2) and can be reflected within the EXI grammar. To create such a grammar, we follow processing steps shown in Fig. 4. That is, we assume a WoT device operates in a concrete context that is described by an ontology. Based on its content, an XML Schema definition is constructed. The schema reflects the RDF/XML basic structure, as well as possible property structures with data type definitions. Following our running example, we will have a class *sensor* in the ontology. A typical individual triple can be (temperature, type, sensor). Thus, the RDF/XML Schema definition will embed and define the *type* element within the *Description* declaration. This, in turn, via the complexType *ClassType*, will declare a list of possible object value assignments for the attribute *resource* (see type *classes*) that are based on all known classes of the (device) ontology (sensor, device, etc.). Based on such an XML Schema definition, an EXI grammar is constructed. The encoding and decoding mechanism of the grammar is described by the EXI standard [17]. The grammar includes an ID resolution (an integer value) of all element/attribute names and possible object value assignments (e.g., sensor). More precisely, each state will have a unique EXI ID representation, which can be used to identify particular elements/attributes. Thus the usage of the string-based representation is not further required.

The semantic repository will use these IDs to represent triples. This will be true for the properties and, if so, string-based values. E.g., the EXI snipped

00 'temperature' 0 ...

will be represented as (temperature, 3, 0) in the repository (see Fig. 3 and Sect. 3.2). Thereby, the temperature value will be saved as a string, the property entry *type* is represented by the EXI ID 3, and the object value is represented as

the EXI enumeration value 0. Note that an unknown subject value will be only saved once in the repository. For a triple with a known subject, it will be only required to save its new property and the value. The subject itself will be linked to an existing subject entry.

An ID manager as used in the generic approach is not necessary since we do not have to put the effort in synchronization of string tables with the semantic repository (to prevent inconsistency to occur when a deletion is requested). This is the benefit of having EXI IDs, that are fixed by the pre-determination of the EXI grammar and based on the XML Schema which was structured based on a (device) ontology. Consequently, this will also reduce the number of strings in the RDF-based serialized messages. The downside of this approach is that the requester of the uRDF Store service needs the same EXI grammar to read the (result) RDF graph message.

It should be also noted that a complex ontology can, at the same time, lead to complex XML Schema, and respectively to complex EXI grammars. To tackle this challenge, in [14] we have presented an approach where we can further optimize the EXI grammar. The technique is called the context-based grammar optimization. It removes EXI grammar fragments (states and transitions) that are not needed for a particular context implementation. For example, an ontology that is defined for building automation scenarios will contain information, among others, such as for sensors and actuators. A constrained embedded device, that operates only a temperature and a humidity sensor, do not really need the EXI grammar definition that is intended to actuator-based devices. Consequentially, these grammar fragments can be omitted then.

5 Evaluation

EXI is known to reach a high compression rate and to be a very fast coding mechanism at the same time (see e.g., [2] and [14]). In this section we will evaluate the applicability of the presented μRDF Store. An RDF/XML-based serialization/de-serialization with the standardized EXI format will be presented. Tests are run in the embedded domain. We will focus on the compactness of representation of RDF-based documents, and required memory. Both generic and concrete grammars will be considered.

5.1 Dataset, Target Platform, and Implementation

As a dataset we use an ontology which was motivated by scenarios from a public-funded ITEA Project, called 'Building as a Service' (BaaS)[10]. Among others, this ontology defines concepts related to sensors in a building (extended from the W3C Semantic Sensor Network (SSN) ontology [16]), as well as concepts related to locations (e.g., location of a sensor in a building). Our triple sets are relatively small (20, 40, 60, 80, and 100 triples). This is justified with the fact that a tiny device such as a temperature sensor can be described with such a moderate semantic description, and has no resources to store more triples.

[10] http://www.baas-itea2.eu.

As a target platform we have selected the well known ARM Cortex-M3 microcontroller with following specification: 72 MHz CPU, 64 kBytes of RAM, and 256 kBytes of Flash memory.

Our generic μRDF Store implementation is written in C and uses a CoAP-based Web service interface for interaction with the store. For our μRDF Store that operates on concrete grammars, we have used our code generation approach [14]. To evaluate the alternative serialization formats, we have used implementations from the HDT and Thrift approaches (see Sect. 2).

5.2 Compactness

Figure 5 shows the resulting size as a percentage of the original plain-text RDF/XML document size (= 100 %). The diagram compares the RDF serialization in following formats: EXI generic, EXI concrete, HDT, and Thrift. The X-axis represents the number of triples that were serialized, and shows the document size (in bytes) when the triples are represented in a plain-text RDF/XML format. As can be seen, EXI serializations reach the best compactness results in all test cases. Especially, the approach is very effective if we use the concrete grammar for the RDF serialization. For this case, we have achieved results which are up to 15 times smaller than the equivalent RDF/XML representation. This is one of the key strengths of the EXI-based approach, which also in a concrete deployment leads to a less network traffic. Furthermore, the opportunity arises to pack a complete message in one or only few data packages that are provided by a constrained network protocol. For instance, the IETF IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) [1] protocol provides only a payload between 81 and 102 bytes (depending on the configuration). Consequently, to transport 20 triples the equivalent concrete EXI representation (=78 bytes) would need only one 6LoWPAN package.

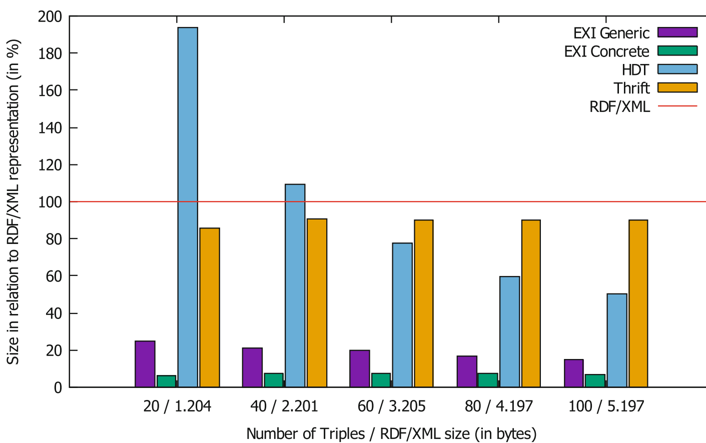

Fig. 5. RDF serialization size

Table 1. Memory usage on ARM Cortex-M3 (in Bytes)

	Generic		Concrete	
#Triples	ROM	RAM	ROM	RAM
20	36521	3422	31957	2398
40	36521	4958	31957	2654
60	36521	6366	31957	2910
80	36521	7902	31957	3294
100	36521	9310	31957	3550

Because of the dictionary that is carried in each RDF document (see Sect. 2), HDT will perform better than RDF/XML when the number of triples is (relatively) big. In our case this is true starting with the triple set of 60. Comparing the 100 triple test set, HDT is still 3 times bigger compared to the generic approach and 7 times bigger compared to the concrete variant. The serialization results of Thrift are always better then RDF/XML and better than HDT in cases of data sets with 20 and 40 triple. However, they are still not as good as results with the EXI serializations (in average 7 times bigger compared to the generic, and 12 times bigger compared to the concrete EXI approach).

5.3 Code Footprint

An important property in order to successfully realize a semantic repository is the memory usage. As already mentioned, constrained embedded devices such as microcontrollers are heavily restricted in this issue.

For evaluating the memory usage we compiled the μRDF Store as presented in Sect. 4 for an ARM Cortex-M3 microcontroller. Table 1 shows the result of the ROM and RAM usage compiled for the different amount of triples (20, 40, 60, 80, and 100) and μRDF Store variants (basic and concrete). Even though the generic μRDF Store variant is able to keep as many triples as RAM is available on the microcontroller, we defined an upper limit of triples which are to be expected (or really needed) for a particular use case. This makes it also easier to compare both variants in terms of memory usage.

The code footprint (ROM) of the concrete variant is around 5 KBytes less compared to the generic variant. This is the advantage of not having implemented an ID manager and the string tables that are needed by the generic variants. Here, the code footprint (ROM) of the generic and concrete μRDF Store is for all triple test cases the same. This is based on the fact that both operate always on their same EXI grammar: The generic one operates on the generic RDF framework grammar and the concrete variant on the grammar that was derived from the ontology and XML Schema respectively (see Sect. 4.2).

The main difference is in the RAM size. The most impact can be seen at the generic approach where the used RAM usage increases rapidly in relation to the number of used triples. This is justified by the needed string tables that are

used by EXI to keep unknown element/attribute names and string-based object values to assign an EXI ID which is then used in the semantic repository. The concrete μRDF Store is able to use and operate directly on the EXI IDs since all possible variants are already pre-determined at the time of grammar generation. Consequently, less RAM has to be used, e.g., to save 100 triple in the repository. Compared to the generic variant, almost 3 times more RAM usage is needed to manage the same triple set.

5.4 Summary

The evaluation results showed that we can significantly reduce the RDF representation size by applying the EXI format as serialization format. Furthermore, this approach is also efficiently applicable to constrained devices such as microcontrollers for implementing, e.g., a semantic repository.

6 Conclusion and Future Work

The W3C standard RDF serialization formats incur high cost in parsing, processing and storing RDF data. This issue becomes especially apparent when RDF data needs to be handled by constrained embedded devices, and it significantly hinders usage of Semantic Technologies in the domain of Web of Things applications. In this paper we have proposed an approach to tackle this issue. The approach is based on the Efficient XML Interchange (EXI) Format – a W3C standard binary format for XML. We have adapted the EXI approach to make it applicable for RDF too. We have proposed a generic RDF EXI grammar, as well as an application specific one. We have implemented the proposed grammars and compared our implementation to state of the art implementations. It is worth noting that our approach is not only efficient but also established on a W3C standard, which is an important feature when it comes to the deployment of the technology in industry settings.

In the future, we will enable RDF EXI data to be queried with a SPARQL-like language and reasoned with an inference engine, adapted for the embedded domain.

References

1. Bormann, C., Mulligan, G.: Ipv6 over low power wpan (6lowpan) (2009). http://datatracker.ietf.org/wg/6lowpan/charter/
2. Bournez, C.: Efficient XML Interchange Evaluation. W3C Working Draft, 7 April 2009. http://www.w3.org/TR/exi-evaluation/
3. Cleary, J.G., Witten, I.H.: Data compression using adaptive coding and partial string matching. IEEE Trans. Commun. **32**(4), 396–402 (1984)
4. Fablet, Y., Peintner, D.: Efficient XML Interchange (EXI) Profile for limiting usage of dynamic memory. W3C Recommendation, 09 September 2014. http://www.w3.org/TR/exi-profile/

5. Faye, D.C., Curé, O., Blin, G.: A survey of RDF storage approaches. Revue Africaine de la Recherche en Informatique et Mathématiques Appliquées **15**(1), 25 (2012)
6. Fernández, J.D., Llaves, A., Corcho, O.: Efficient RDF interchange (ERI) format for RDF data streams. In: Mika, P., Tudorache, T., Bernstein, A., Welty, C., Knoblock, C., Vrandečić, D., Groth, P., Noy, N., Janowicz, K., Goble, C. (eds.) ISWC 2014, Part II. LNCS, vol. 8797, pp. 244–259. Springer, Heidelberg (2014)
7. Fernández, J.D., Martínez-Prieto, M.A., Gutiérrez, C., Polleres, A., Arias, M.: Binary RDF representation for publication and exchange. Web Semant.: Sci., Serv. Agents World Wide Web **19**, 22–41 (2013)
8. Fernández, N., Arias, J., Sánchez, L., Fuentes-Lorenzo, D., Corcho, Ó.: RDSZ: an approach for lossless RDF stream compression. In: Presutti, V., d'Amato, C., Gandon, F., d'Aquin, M., Staab, S., Tordai, A. (eds.) ESWC 2014. LNCS, vol. 8465, pp. 52–67. Springer, Heidelberg (2014)
9. Goldman, O., Lenkov, D.: XML binary characterization. World Wide Web Consortium, Note NOTE-xbc-characterization-20050331, March 2005. http://www.w3.org/TR/2005/NOTE-xbc-characterization-20050331
10. Hasemann, H., Kröller, A., Pagel, M.: RDF provisioning for the internet of things. In: 3rd IEEE International Conference on the Internet of Things, IOT 2012, Wuxi, Jiangsu Province, China, October 24–26, 2012, pp. 143–150 (2012)
11. Hasemann, H., Kröller, A., Pagel, M.: The wiselib tuplestore: a modular RDF database for the internet of things. CoRR, abs/1402.7228 (2014)
12. Huffman, D.A.: A method for the construction of minimum-redundancy codes. Proc. Inst. Radio Eng. **40**(9), 1098–1101 (1952)
13. ISO/IEC. Iso/iec dis 15118–2: Road vehicles - vehicle to grid communication interface - part 2: Network and application protocol requirements (2012)
14. Käbisch, S., Peintner, D., Heuer, J., Kosch, H.: Optimized XML-based web service generation for service communication in restricted embedded environments. In: 16th IEEE International Conference on Emerging Technologies and Factory Automation (2011)
15. Katz, P.W.: String searcher, and compressor using same, September 24 1991. US Patent 5,051,745
16. Henson, C., Lefort, L., Taylor, K.: W3c semantic sensor network incubator group (SSN-XG). W3C XG, W3C (2011). http://www.w3.org/2005/Incubator/ssn/
17. Schneider, J., Kamiya, T., Peintner, D., Kyusakov, R.: Efficient XML Interchange (EXI) Format 1.0 (Second Edition). W3C Recommendation, W3C, February 2014. http://www.w3.org/TR/2014/REC-exi-20140211/
18. Shelby, Z., Hartke, K., Bormann, C.: Constrained application protocol (coap). Technical report, IETF (2013). http://datatracker.ietf.org/doc/draft-ietf-core-coap/
19. W3C. RDF 1.1 Concepts and Abstract Syntax (2014). http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/
20. W3C. RDF 1.1 XML syntax (2014)
21. ZigBee. Smart Energy Profile 2 (SEP 2) (2013). http://www.zigbee.org/