The Semantic Web in an SMS

Onno Valkering, Victor de Boer, Gossa Lô,
Romy Blankendaal, and Stefan Schlobach

Vrije Universiteit Amsterdam, the Netherlands
o.a.b.valkering@student.vu.nl, v.de.boer@vu.nl, a.g.lo@vu.nl,
r.a.m.blankendaal@student.vu.nl, k.s.schlobach@vu.nl

Abstract. Many ICT applications and services, including those from the Semantic Web, rely on the Web for the exchange of data. This includes expensive server and network infrastructures. Most rural areas of developing countries are not reached by the Web and its possibilities, while at the same time the ability to share knowledge has been identified as a key enabler for development. To make widespread knowledge sharing possible in these rural areas, the notion of the Web has to be downscaled based on the specific low-resource infrastructure in place. In this paper, we introduce SPARQL over SMS, a solution for Web-like exchange of RDF data over cellular networks in which HTTP is substituted by SMS. We motivate and validate this through two use cases in West Africa. We present the design and implementation of the solution, along with a data compression method that combines generic compression strategies and strategies that use Semantic Web specific features to reduce the size of RDF before it is transferred over the low-bandwidth cellular network.

1 Introduction

The Semantic Web by design builds on, and relies on, the Web infrastructure for data exchange. This includes sophisticated server and network infrastructures which are unavailable in many rural areas of developing countries. These areas are not reached by the web and its possibilities while at the same time the ability to share knowledge has been identified as a key enabler for development. To make knowledge sharing possible in rural developing areas, the notion of the Web has to be *downscaled* based on the specific low-resource infrastructure in place [6].

Data sharing solutions, such as those based on Semantic Web and Linked Data technologies, should not only be accessible to those with abundant resources and reliable infrastructures, but also in low-resource environments. The flexible graph models of the Semantic Web and its language-agnostic nature make it especially useful for data sharing in these location, because of the many different spoken languages and customs. In [2] we show that locally produced market data, stored as RDF, is produced through, and used in, a voice-interface accessible for low-literate users in their preferred language. We also identified opportunities for data sharing and integration. More recently, we developed the Kasadaka[1], a low-resource prototyping and computing platform which uses semantic technologies specialized at developing multi-modal user interfaces, e.g. touchscreens or voice, for low literacy in rural areas of development countries. While this widens the applicability and possible use-cases, a core problem remains; the lack of infrastructure for Web-like sharing of information in the targeted rural areas.

The main challenge lies in the unavailability of network connections. Especially in many rural areas of developing countries, internet connections are either missing or extremely unreliable. Internet penetration is estimated to be 28.6% of the population in Africa as a whole (compared to 52.8% in the rest of the world), with some countries reaching considerably less of their population: for example 7.0% in Mali[2]. These numbers include both urban and rural areas and in the latter, internet penetration is virtually non-existent. The Semantic Web is built on top of the Internet (TCP/IP) and Web infrastructure (including HTTP) and as such when no Internet is available, it is unusable. However, we can design solutions to implement Web-like data sharing using alternative networking capabilities available in low-resource environments.

We present a specific downscaling solution for exchanging (RDF) data in which HTTP is substituted by SMS to enable Web-like exchange of data over cellular networks. We show the viability of this solution in two different ways:

1. Technological: we identify three main technological problems when using an SMS protocol as semantic data transfer protocol, message size, the asynchronous nature of the protocol and how to deal with pagination issues. Our solution is validated w.r.t. each of those problems with a variety of methods, which includes a large-scale empirical comparison of compression size.
2. Societal: Using two use-cases from Sub-Saharan Africa (one from Ghana, one from Mali) we will show how SMS-based Semantic Web can practically address the knowledge sharing needs of rural communities. We introduce these in Section 3 and validate our solution against these cases in Section 6.

While in this paper, we present a practical solution to low-bandwidth knowledge sharing, our investigation will also be more generally useful to understand how Semantic Web principles and practices can be separated from the infrastructure layers that often are assumed to be prerequisites.

2 Related Work

SMS as a data channel has been proposed in other ICT for Development (ICT4D) cases, for example in [8]. Mobile banking -including through SMS- has been well-established in many developing economies (cf. [10]). A number of Social Network Services such as Twitter, Facebook as well as the Google search engine allow for accessing those services through SMS[3]. Mostly, this deals with machine-to-human interaction and not, as in our case for machine-to-machine (M2M)

[1] `http://www.kasadaka.com`. “Kasadaka” roughly translates to “Talking Box” in a number of Ghanaian languages

[2] as of November 2015 `http://www.internetworldstats.com/`

[3] `http://www.digitaltrends.com/mobile/sms-your-way-back-to-the-web/`

interaction. Related work in semantic data exchange in low-resource network environments includes the Entity Registry System (ERS) [3], an open-source entity registry specifically designed for environments with ad-hoc and/or unreliable network connectivity, as is often the case in rural areas. It allows for Linked Data without using the centralised components that make up the Web infrastructure. ERS has mechanisms to deal with interval-based network connectivity (e.g. a mobile truck that functions as an access point) and is resistant against packet loss. DakNet provides similar solutions where ad-hoc wireless networks are combined with asynchronous networking, also including mobile access points [11]. Whereas these solutions also implement Web-like data exchange without Web infrastructure, they focus mainly on local networks and rely on the availability of partial Internet connectivity.

Another way of transferring data without Internet is through so-called Sneakernets, where data is exchanged by physically moving removable media or hard disks. For large-scale non-immediate data transfer, this is a viable solution [5] which can be combined with solutions such as the one presented in this paper.

In this paper, we focus on semantic data exchange using the SPARQL protocol. There are other opportunities for accessing RDF data over a network. Two examples are simple URI dereferencing and the use of Linked Data Fragments [13]. Compared to these methods, accessing RDF using SPARQL typically takes more computing resources on the client and server devices itself, but allows for more fine-grained querying by which bandwidth can be limited. For our specific cases, saving bandwidth is a key issue, which is why we use SPARQL. It is interesting to further investigate the trade-off between computational and networking resources in these specific ICT4D cases.

3 Information Sharing in the Absence of the Web

As early as 2011 we pointed to some negative effects of the Semantic Web's reliance on Web infrastructure [6], which effectively made this technology inaccessible for a majority of the world population. Through a number of research projects in Sub-Saharan Africa we have since then identified numerous use cases that rely on knowledge sharing. The recent Kasadaka project builds on information acquired in Burkina Faso, Mali, Ghana and Niger and aims at providing information to people living in rural communities for several different use-cases. It provides a generic platform, which enables voice- and SMS-based communication over GSM and can be deployed in communities and owned and maintained by local stakeholders. The platform can host different information sharing services, accessible through simple icon-based visual interfaces or voice interfaces callable from any mobile phone as users especially those in remote rural villages are often low-literate and speak local languages.We regularly ran into conceptual and technical problems for which the Knowledge Engineering community has already provided robust methods, as most real use cases require data and knowledge sharing across communities and devices. We here describe two cases that have been co-developed with local stakeholders in rural West Africa.

3.1 The DigiVet Case

One of the information needs identified for and by rural farmers is on animal health, in particular on diagnosing animals. DigiVet is a voice-based veterinary information service that support subsistence farmers in making the decision whether or not to visit a veterinarian. Animal diseases spread within and between villages and can often be cured merely with the intervention of a veterinarian. The problem that arises in these rural areas is that local expertise is often lacking and poor infrastructures (poor roads, lack of electricity) prevent access to information and sharing of knowledge. Farmers need information on animal diseases, disease patterns, diagnosis and symptoms to take preventive action and preclude cattle loss, but this cannot be shared easily over large distances.

DigiVet includes a simple interface which presents farmers with a set of symptom related questions on a touchscreen. It is based around a knowledge base[4] developed by interviewing veterinarians working in rural Northern Ghana. The system provides a diagnosis whether or not a farmer should contact a veterinarian. DigiVet relies on semantic data exchange between farmers and veterinarians at large distances. While the used Kasadaka platform is suitable for the creating the interface for diagnosing, there is currently no technology to cater the necessary semantic data exchange.

3.2 The RadioMarché Case

The RadioMarché case, introduced in [1], is a market information system designed to gather and distribute information about offerings of specific produce on local markets in the Tominian region of Mali. To allow low-literate stakeholders to retrieve market information in the absence of internet connection, a voice-accessible service was built that can be called from any mobile phone. The service can be called by local farmers in their own language to retrieve this information. Community radio hosts retrieve and broadcast local offerings on the radio. The system was developed and deployed in 2012 [7]. The gathered product offering data was ported to the RDF data model and a Semantic Web compliant version was developed. The benefits of linking market data to external data sources and using this for visualization and improved data analyses, in particular for stakeholders such as NGOs or bulk buyers is described in [2].

4 A Platform for Semantic Web in an SMS

Our goal is to make the use of Semantic Web applications possible in areas lacking an infrastructure to support Web-like exchange of data. The intent is not to create an isolated network that mimics Semantic Web practices, but rather to develop a mechanism that supports the retrieval and manipulation of RDF data across different kind of networks infrastructures. We want to achieve this without imposing additional network-specific operations for application developers. This

[4] `https://github.com/biktorrr/digivetkb`

means that applications can still be developed for HTTP, the mechanism sits in between, converting messages to be able to cross the specific low-bandwidth network in place, without requiring the application to be adjusted.

4.1 SPARQL over SMS

The rural areas in West Africa targeted by our use cases only have cellular networks available for digital communication. Applications deployed in these areas can make use of SMS for M2M transfer of data, instead of HTTP as part of a Web-based network. Noteworthy practical differences between SMS-based networks and Web-based networks are:

– SMS-based network agents are identified by phone numbers instead of URLs;
– The size of an SMS is limited up to 160 bytes[5];
– SMS implements a one-way messaging pattern, whereas HTTP implements a request-response messaging pattern.

To transfer HTTP messages, produced by Semantic Web applications, over a SMS-based networks, a conversion mechanism is required. This mechanism, in addition to the above-mentioned differences, must take into account these case-specific requirements:

– the number of messages sent should be as low as possible, in view of costs;
– the mechanism should be possible to run on affordable hardware[6].

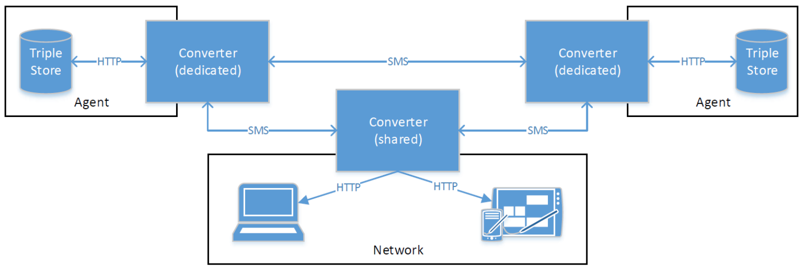
Although the costs per transferred byte are relatively high for SMSes, it builds on existing infrastructure which has a global reach including many rural areas of development countries. Also, the required hardware to be able to send SMSes is affordable and widely available.

Our implementation of the described mechanism is called *SPARQL over SMS*. By supporting the CONSTRUCT and INSERT/DELETE DATA query forms a basic usage of Semantic Web for M2M communication is realized. We select this subset of SPARQL as it involves simple data transfer using RDF triples. SELECT query responses (where they are not part of a CONSTRUCT query) take the form of result tables of arbitrary sizes and are harder to optimize.

Figure 1 gives an overview of SPARQL over SMS. In the context of SPARQL over SMS, application that can both send and receive SPARQL queries are called *agents*. The *converter* is a key component responsible for the conversion between an HTTP SPARQL request and an SMS-optimized equivalent. Different options for sending and receiving SMSes are supported by the converter, such as a GSM dongle or an online SMS service. This allows the converter to be deployed in various scenarios. A converter deployed in a data-center could be used to share data with a triple store running on low-resource hardware deployed in the field. This can be useful when aggregation of data from multiple devices is desirable.

[5] Based on the encoding used: 8-bit supports 140 characters, 7-bit up to 160 characters.
[6] Such as a Raspberry Pi computer: `https://www.raspberrypi.org`.

Fig. 1. SPARQL over SMS Overview



A converter instance supports two modes: shared and dedicated. A dedicated converter maps a phone number directly to a particular agent, making it possible for the agent to both send and receive SPARQL queries. Shared converters can serve multiple agents, so that a phone number (assigned to a single GSM dongle) cannot be mapped to a single agent. In this case, the agents can only perform outgoing SPARQL queries but cannot be the target of incoming queries. Sending SPARQL queries requires an endpoint URL identifying the target receiver. To allow the targeting of an agent in an SMS-network, the converter can provide a URL representation of an arbitrary phone number. For example, the format of a SPARQL endpoint URL is: *http://{converter hostname}/agent/{phone number}/sparql*. SPARQL requests sent to such an endpoint are captured by the converter and sent to the phone number. The converter then receives the query and runs it on the triple store of the associated agent. The result is then send back to the initial converter which returns is as the response to the SPARQL request in the specified format.

4.2 SMS message structure and conversion

SMSes sent between converters follow a specific structure. Five characters of each SMS are reserved for metadata for which the basic 7-bit character set, as specific by the GSM 03.38 character set[7], is used. This includes the message type, message identifier, and position for multi-part messages. Excluding the non-print characters there are 125 different characters left that can be used. A single character can thus express a numerical value of 1 to and including 125.

HTTP to SMS The converter creates optimized representations of HTTP SPARQL queries and results. In the case of a SPARQL query the encoding routine is based on the SPARQL query form to perform fine-grained optimizations. The compact representation is optionally split into multiple SMSes if it exceeds the character limitation of a single SMS. The position of each part will be indicated by the multi-part position character in the metadata. After conversion, the compact representation is send over SMS to the phone number extracted from the endpoint URL used to send the SPARQL query to the converter.

[7] `http://www.3gpp.org/DynaReport/23038.htm`

SMS to HTTP When a converter receives an SMS, HTTP representations are reconstructed. As different encoding routines might be used, the appropriate routine is based on the message type defined in the metadata. Decoding cannot guarantee a result exactly identical to the original message. The resulting message might thus not be syntactically equivalent, but it will be semantically equivalent. Multi-part messages are concatenated based on the multi-part position in the SMS metadata.

5 Research Challenges

In the design of our solution, converters are used to transfer SPARQL queries and results between Web- and SMS-based networks. However, to develop a workable solution, a number of challenges need to be addressed. In this section we outline the different challenges, namely: how to reduce the size of RDF data to allow for efficient transfer over SMS, issues around asynchronicity of communication and how to deal with unpredictable query result sizes.

5.1 Small RDF Data Compression

The serialization format has a great effect on the size of an RDF file, and thereby on the amount of SMSes needed to transfer the data. The costs associated with SMSes restricts us to cases with small amounts of triples which. Still, to save costs associated with the sending SMSes, we want to use the combination of RDF serialization and compression that is most efficient, in terms of transfer size, for such small RDF data sets.

Experimental Setup To identify the best serialization and compression combination we run experiments on RDF data sets provided by the LOD Laundromat [12]. These RDF files are crawled from multiple Linked Data sources, making it a realistic representation of real-world RDF data sets. Our benchmark consists of 232,822 RDF files with size between 1 and 1000 triples. This large-scale experiment ensures that we test across many types of data sets and various characteristics which might influence serialization and compression.

The files were converted to various serializations (RDF/XML, Turtle, HDT and EXI). RDF/XML and Turtle are plain text serialization formats specifically designed for RDF data. The binary format "Header, Dictionary, Triples" (HDT) is a data structure developed to compactly store and exchange RDF data without sacrificing the ability to query the data [4]. Efficient XML Interchange (EXI) is a binary format designed to create compact representations of XML and has been proposed for efficient RDF exchange in constrained embedded networks [9]. For each format, including the original N-Triples format, the file size is recorded before and after applying gzip compression. The default implementations of RDFLib[8], HDT[9] and EXIficient[10] have been used.

[8] `https://github.com/RDFLib/rdflib`
[9] `https://github.com/rdfhdt/hdt-java`
[10] `https://github.com/EXIficient/exificient`

Results Table 1 lists per format the average file size w.r.t to the original N-Triples format. The best compression per bin is marked bold. The binary gzip, HDT and EXI formats include Base64 encoding overhead. We first look at the results from the RDF files in the range of 1 to 100 triples (81,492 in total) in bins of 10. As expected, the size reduction compared to the original format increases with the number of triples, due to more syntactic redundancy. HDTs are bigger than the original due to the HDT's metadata. With files up to 30 gzip compressed N-Triples outperforms the other formats. Above 50 triples, the compressed Turtle format outperforms compressed N-Triples. For sets between 30-60 triples, the uncompressed EXI format performs similar to compressed N-Triples and compressed Turtle. Applying gzip compression to EXI hardly has any effect and even increases the file size in most cases.

When considering RDF files between 100-1000 triples we note that gzip compressed Turtle results in the best compression. In addition, RDF/XML stagnates around 51% and Turtle around 36% of size compared to N-Triples. For files with 600+ triples gzip compressed HDT drops below the size of EXI, at the cost of losing the ability to directly query the HDT files due to an additional layer of gzip compression. We conclude that for the smallest data sets ($\leq$40 triples), gzip compressed N-Triples is preferable. For data sets between 40-1000 triples, gzip compressed Turtle serialization scores best. The reason N-Triples performs better than Turtle for the smallest data sets can be the added overhead of prefixes in Turtle[11]. In our implementation, we decided to dynamically select the appropriate serialization (N-Triples or Turtle) based on the number of triples in the SPARQL result.

Table 1. Results of the LOD Lab compression experiment (N-Triples = 100%).

No. Triples	N-Triples +Gzip	RDF/ XML	RDF/ XML +Gzip	Turtle	Turtle +Gzip	HDT	HDT +Gzip	EXI	EXI +Gzip	Comb. method
1-10	**50.7**	103.8	77.0	102.0	70.3	495.5	180.1	57.5	65.9	44.2
11-20	**22.5**	62.0	27.1	50.5	24.2	122.2	47.0	23.3	24.9	18.9
21-30	**16.2**	58.2	18.5	48.7	16.3	79.5	31.1	16.5	17.5	13.6
31-40	28.3	69.1	30.9	62.1	28.6	86.5	40.7	**28.2**	29.1	23.5
41-50	9.8	51.2	10.2	42.3	**8.6**	38.1	14.8	9.3	9.7	8.0
51-60	17.2	59.2	17.5	50.1	15.9	50.5	22.8	**15.8**	16.3	8.7
61-70	11.8	58.5	12.4	42.4	**10.0**	43.0	17.7	11.1	11.6	6.0
71-80	8.8	54.8	8.5	40.9	**7.0**	31.6	11.2	7.5	7.8	6.4
81-90	6.7	52.0	6.3	40.6	**5.1**	25.4	9.1	5.8	6.0	4.4
91-100	8.1	54.9	7.6	40.4	**6.2**	26.9	9.7	6.8	7.0	5.7
101-200	8.8	62.0	8.3	39.2	**6.7**	24.7	10.1	7.6	7.9	5.7
201-300	4.8	50.8	3.6	39.0	**2.8**	13.4	4.0	3.6	3.6	2.7
301-400	4.8	51.5	3.3	37.7	**2.5**	11.4	3.3	3.0	3.1	2.5
401-500	4.4	51.5	2.9	37.4	**2.2**	10.4	2.7	2.6	2.7	2.2
501-600	5.0	53.8	3.4	38.7	**2.5**	8.9	3.0	2.9	3.0	2.4
601-700	4.1	51.0	2.5	35.9	**1.7**	8.5	2.2	2.3	2.4	1.7
701-800	4.5	51.1	2.7	36.2	**1.9**	8.1	2.1	2.4	2.4	1.9
801-900	4.4	51.1	2.6	36.4	**1.8**	7.9	1.9	2.3	2.3	1.8
901-1000	4.1	50.9	2.4	36.5	**1.7**	7.7	**1.7**	2.1	2.1	1.7

[11] The used Turtle serializer adds RDF, RDFS, XSD and XML prefixes by default.

5.2 Shared Vocabulary/Semantic RDF Data Compression

Section 5.1 focused on serialization and gzip compression. These generic strategies consider only the syntactical representation of RDF. In order to reduce the size of RDF data even more, we also tested two compression strategies focused on RDF content-specific aspects. We do this on the basis of RDF vocabularies that define reusable definitions for common properties and/or types.

Experiment Setup We experimented with 30 popular vocabularies[12]. To 3,577 RDF data sets from the previous experiment dictionary-encoding and reasoning, based on the RDF vocabularies, was applied. The experiment had three rounds with an increasing number of vocabularies (most popular 10, 20 and 30). A single HDT file, containing combinations of vocabularies, is generated in each round, for which the dictionary-component is used as dictionary encoding. During the encoding all the URIs that occur in one of the vocabularies are replaced with a placeholder containing the identifier generated by HDT for an URI.

We use reasoning to find semantic redundancies in RDF data sets, based on the vocabularies. The implemented reasoner searches for redundancies based on twelve RDFS entailment patterns[13], as well as rules for two OWL properties: *owl:SymmetricProperty* and *owl:inverseOf*. Semantically redundant triples are removed from the data set. Only explicitly defined triples from the RDF data set and vocabularies are considered. Therefore, it is not guaranteed that the final result is always the smallest set of triples possible.

Results The two compression strategies are measured independently during the experiment. Precondition for these compression strategies is that the subjected RDF data set must use one of the considered vocabularies, which makes that these compression strategies do not always have a size reducing effect.

Size reduction averages have been calculated only for results that led to an actual size reduction, grouped by number of triples in bins of 100. Based on these averages we have found that reasoning based on the top 10 vocabularies has minimal effect, up to 3% average size reduction. Using an additional 10 vocabularies increases the average size reduction across all bins, resulting in average size reduction ranging from 8.7% to 13.4%. Using the top 30 vocabularies has no advantage over the top 20 vocabularies when using reasoning. The dictionary-encoding achieves around 6.5% average size reduction based on the top 10 vocabularies. This is slightly improved to around 8% when using the top 20 vocabularies. An additional, but minimal improvement, can be obtained when using all the 30 vocabularies for dictionary-encoding. Furthermore, it stands out that the dictionary-encoding could be used more consistently, 96% of the files could be reduced by using dictionary-encoding against 31% for reasoning. Based on these results we conclude that it is best to use the top 20 popular vocabularies when using the reasoning and dictionary-encoding compression strategies. The minimal improvement of using the top 30 for dictionary-encoding is not commensurate to the increase of processing duration and maintenance efforts introduced by the additional 10 vocabularies.

[12] Including YAGO, FOAF, and SKOS. Based on `http://prefix.cc/popular/all`.

[13] `https://www.w3.org/TR/rdf11-mt/#rdfs-entailment`

We combined vocabulary based compression strategies with the generic compression strategies from Section 5.1 to form a RDF compression method for SPARQL over SMS. The LOD Lab data sets, as described in Section 5.1, have been subjected to this combined method to measure the performance. The results are listed in Table 1. It shows that the added reasoning and dictionary-encoding strategies are especially effective when compressing the smallest RDF data sets (1-200 triples). As the number of triples increases, the syntactical compression strategies become more efficient and gradually make the vocabulary based compression strategies less beneficial. This is also follows from Table 2, the added vocabulary based compression provides a head-start in terms of the average number of triples that can be sent per SMS. This holds up to 10 SMSes.

Table 2. Average number of triples that can be send based on the number of SMSes.

Nr. of SMSes	Only serialization and compression	Added shared vocabulary compression
1	0	0
2	3	3
3	6	8
4	9	16
5	21	24
6	66	84
7	84	98
8	116	126
9	175	189
10	301	301

5.3 Blending Synchronous and Asynchronous Messaging

SPARQL exchanges follow a request-response messaging pattern. When a query is sent as a request over the network, the receiver processes the request and composes a response with the query result. A single connection is used and kept open during the transfer, making it a synchronous operation. Sending SMSes follows a one-way messaging pattern. Each message is a standalone message that not enforces a follow-up response. As the connection is terminated after a message has been delivered sending of SMS is an asynchronous operation.

With SPARQL over SMS we want to seamlessly transfer messages from Web-based networks to SMS-based networks and vice-versa. For this purpose, we need to harmonize the two different messaging patterns. The initial implementation of SPARQL over SMS keeps HTTP connections open during the data transfer. After sending a SPARQL query over SMS, the converter waits until the corresponding result response comes in. The query result response is of a different message type, but it can be correlated to the original request by using the message identifier for correlation.

This implementation is functional, but might not be a optimal considering the deployment in rural areas described in our use cases. For example, due to temporary loss of connectivity the response message might be available after hours or even days. It is questionable if the low-resource hardware can hold open multiple connections for that period. Even if it is capable of doing so, there is a genuine risk of a sudden power outage that will result in a loss of all open connections requiring retries. In our specific use-cases this would result in additional, unnecessary, costs. Additional efforts are required to create an asynchronous-supporting version of SPARQL for situations when a response is not expected within a seconds- or minutes-long time span.

5.4 Unpredictable Query Result Sizes

A simple looking SPARQL query might yield an unexpectedly large result. To be thrifty with sending SMSes, we want to restrict the amount of SMSes that will be send. We have considered two options to achieve this, SPARQL pagination and pagination on a SMS level. SPARQL provides the LIMIT, OFFSET and ORDER BY keywords that can be used to implement pagination, but the SPARQL query result will not include pagination information, e.g. the total number of results available. This means it is not possible to tell if all results have been retrieved yet. Another issue is the possibility for a triple to have a very long literal object that can span multiple SMSes. Relying only on SPARQL pagination for regulating the result size is not sufficient to regulate the number of SMSes send. Pagination on the SMS level would only introduce the option to decide whether or not to continue receiving the SMSes. Since a partial result from a complete SPARQL result cannot be created: it is the whole SPARQL result or nothing. This would also alter the way SPARQL over SMS must be used compared to the SPARQL standard, due to the addition of pagination operations which can not be ignored. With our implementation the SPARQL result is directly, after compression, send through SMS. If the result does not fit in a single SMS it will split up the message, on arbitrary points, into multiple SMSes. If a hard set restriction is reached and stops sending SMSes the receiver cannot read the message properly due to missing parts: partial results are not supported. Therefore, we consider above-limit SPARQL results as an error.

6 Practical Validation

6.1 Implementation and Integration

SPARQL over SMS[14] is developed with integration with other services in mind and can be deployed on various operating systems and devices. For validation, we used SPARQL over SMS in combination with Kasadaka[15]. This combination runs on widely available and affordable hardware.

6.2 Evaluation in Four Scenarios

For the two use case in Section 3, we are developing services using Kasadaka running on low-resource devices. The communication between two remotely deployed devices is key. Our setup consists of two Raspberry Pi 2 computers with

[14] `https://github.com/onnovalkering/sparql-over-sms`, available as open source
[15] `https://github.com/abaart/KasaDaka`

both the DigiVet and RadioMarché[16] data sets loaded in a ClioPatria[17] triple store. Four SPARQL queries that correspond to two scenarios per use case are tested to determine the amount of SMSes required both with our RDF compression method and without (plain RDF/XML). To improve the shared vocabulary compression strategies, the use case vocabularies are added to vocabularies used.

Extending the DigiVet application Combining DigiVet with SPARQL over SMS adds new data sharing options. With two new scenarios, from the perspective of the veterinarian, we demonstrate how the DigiVet application can be extended using the new features. First we consider a veterinarian interested in types and frequencies of animal disease symptoms occurring near Walewale, Ghana. The SPARQL query in Listing 1.1 answers this question. Sending the query requires 3 SMSes and yields a result of 7 triples. Returning the result takes 3 SMSes with our solution (without compression it takes 14 SMSes). As a second scenario for the DigiVet use case we consider the need of a veterinarian to update the animal diseases knowledge base present on a DigiVet deployment. As an example, the SPARQL query in Listing 1.2 can be used to add a new disease ("Black Leg") with associated symptoms to the triple store. Transferring this query requires 3 SMS messages (5 without using compression).

Listing 1.1. Digivet SPARQL query for 1st scenario

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dv: <https://w3id.org/w4ra/digivet/>

CONSTRUCT {
  ?sym dv:occurance_count ?count
}
WHERE {
  SELECT ?sym (COUNT(?sym) as ?count) WHERE {
    ?person foaf:based_near <http://sws.geonames.org/2294174/> .
    ?person dv:has_case ?case .
    ?case dv:has_symptom ?sym
  }
  GROUP BY ?sym }
```

Listing 1.2. DigiVet SPARQL query for 2nd scenario

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dv: <https://w3id.org/w4ra/digivet/>

INSERT DATA {
    dv:black_leg a dv:Disease .
    dv:black_leg rdfs:label "Black_leg"@en .
    dv:Cow dv:canCarryDisease dv:black_leg .
    dv:Sheep dv:canCarryDisease dv:black_leg .
    dv:unwillingnessToMove dv:symptom_for_disease dv:black_leg .
    dv:rapidBreathing dv:symptom_for_disease dv:black_leg .
    dv:lameness dv:symptom_for_disease dv:black_leg     .
    dv:appetiteLoss dv:symptom_for_disease dv:black_leg .
    dv:fever dv:symptom_for_disease dv:black_leg .
    dv:swellingThigh dv:symptom_for_disease dv:black_leg .}
```

Extending the RadioMarche application For the RadioMarché service, we consider two scenarios. The first involves the retrieval of the current offerings,

[16] A clone of the store is available at `http://semanticweb.cs.vu.nl/radiomarche/`
[17] `http://cliopatria.swi-prolog.org/`

including the phone number of the advertisers, in the Mafoune and Mandiakuy regions of Mali. Using a CONSTRUCT query (Listing 1.3), this information is retrieved as an RDF graph from a RadioMarché installation. Sending the query through our solution requires 3 SMS messages (4 without using compression). The query result consists of 152 triples in total and could be transferred using 8 SMS messages (121 SMS messages would have been required without compression). This shows the economic impact of the compression step. As a second scenario, we perform an INSERT DATA query (Listing 1.4) to add product labels in more languages. The query creates ten new triples. Our solution requires only 3 SMS messages, half of the uncompressed number.

Listing 1.3. RadioMarché SPARQL query for 1st scenario

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rm: <http://purl.org/collections/w4ra/radiomarche/>

CONSTRUCT {
     ?contact rm:contact_tel ?tel .
     ?contact rm:has_offering ?offering .
     ?offering rdfs:label ?prod_name
} WHERE {
     ?offering a rm:Offering .
     ?offering rm:has_contact ?contact .
     ?offering rm:prod_name ?prod .
     ?prod rdfs:label ?prod_name .
     ?contact rm:contact_tel ?tel .
     ?contact rm:zone ?zone .
     FILTER (?zone IN (rm:zone_Mafoune, rm:zone_Mandiakuy)) }
```

Listing 1.4. RadioMarché SPARQL query for 2nd scenario

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rm: <http://purl.org/collections/w4ra/radiomarche/>

INSERT DATA {
     rm:product-Beurre_de_karite rdfs:label "Shea butter"@en .
     rm:product-Beurre_de_karite rdfs:label "La manteca de karit "@es .
     rm:product-Miel_liquide rdfs:label "Honey"@en .
     rm:product-Miel_liquide rdfs:label "Miel"@es .
     rm:product-Amande_de_karite rdfs:label "Shea nuts"@en .
     rm:product-Amande_de_karite rdfs:label "Nueces de karit "@es .
     rm:product-Tamarin rdfs:label "Tamarind"@en .
     rm:product-Tamarin rdfs:label "Tamarindo"@es .
     rm:product-Graine_de_nere rdfs:label "Nere seeds"@en .
     rm:product-Graine_de_nere rdfs:label "Semillas Nere"@es . }
```

Discussion Table 3 summarizes the results for all scenarios. It shows the number of SMSes needed to transfer the query as well as the query response. For the realistic use cases, the amount of SMS per query is limited. We also list the total costs per query by converting current local SMS rates from two providers to US Dollars[18]. This suggests that, although expensive, the use case could potentially be made economically viable. The number of SMSes required to transfer the SPARQL results confirm to the estimations in Table 2.

[18] For Mali, we assume an average cost of 20CFA=0.035USD per SMS `http://www.orangemali.com/2/particuliers/28/34/les-prepayes-113.html` (accessed April 2016). For Ghana, we assume 0.055GH=0.014USD per SMS `http://support.vodafone.com.gh/customer/portal/articles/1823814-sms` (accessed April 2016)

Table 3. Summary of the four validation scenarios.

Scenario	Location	Query type	Request Size in nr. of SMS	Request est. cost (USD)	Response Size in nr. of SMS	Response est. cost (USD)
Digivet Sc.1	Ghana	CONSTRUCT	3	0.042	3	0.042
Digivet Sc.2	Ghana	INSERT	3	0.042	n.a.	
RadioMarché Sc.1	Mali	CONSTRUCT	3	0.105	8	0.280
RadioMarché Sc.2	Mali	INSERT	3	0.105	n.a.	

7 Conclusions

We show that using the Semantic Web for data sharing is possible in areas without a Web infrastructure. We developed a conversion module that translates SPARQL over HTTP requests to SMSes and decodes these messages at the other end. SPARQL over SMS is an example of downscaling the Semantic Web to the infrastructure in place, in our case SMS. Extending the Kasadaka platform with this M2M communication functionality adds new possibilities for Semantic Web applications. Our solution integrates easily with other data sharing solutions since it does not create an isolated SMS-network but presents a conversion mechanism.

We investigated a number of challenges around porting SPARQL data exchange using SMS. Several RDF compression strategies are evaluated based on real-world small data sets, leading us to a dynamic compression method that combines the generic serialization and text compression strategies with strategies using shared vocabulary. We show the viability of sending small RDF data sets using SPARQL over SMS and elaborate this in four scenarios from two realistic use cases. Future work consist of further development and deployment of solutions which include SPARQL over SMS in the field and designing longer term evaluations for these and new ICT4D use cases.

The current SPARQL over SMS has several limitations and opportunities for improvement. First, the reasoning that is used to eliminate semantic redundancies is based on a limited number of RDFS and OWL patterns and is restricted in terms of the search depth. Second, the SMS transfer mechanism is not yet fitted to properly deal with unexpected faults or partial transfers. We are looking at methods from systems such as the aforementioned ERS. Furthermore, not yet all SPARQL operations are supported. To achieve full compatibility, these will have to be implemented. Lastly, the implementation used to send and receive SMSes only supports 8-bit SMSes (140 characters). Using 7-bit SMSes (160 character) can further increase efficiency. The intent is to conduct further tests, by deploying SPARQL over SMS in the field, to identify the effects of these limitations and to validate the solution in real-world conditions. These field tests will include research into the economic viability of these solutions as discussed in [7] and look, for example, at integrating mobile-based payment plans.

Although SPARQL over SMS is developed based on ICT4D cases, it is applicable to other low-bandwidth cases. For example in the context of disaster-management or Internet of Things. The technologies of SPARQL over SMS are platform independent and it can be ported to other cases and platforms.

Finally, in this paper, we presented a specific approach for decoupling the principles and practices of the Semantic Web from the underlying implementation. This shows that these principles are still valid and valuable without the availability of Internet and a Web infrastructure. The more non-Web-Based networks are supported, the greater the reach of Semantic Web will be, as knowledge can be send across multiple types of networks in a standardized fashion.

References

1. de Boer, V., De Leenheer, P., Bon, A., Gyan, N.B., van Aart, C., Guéret, C., Tuyp, W., Boyera, S., Allen, M., Akkermans, H.: Radiomarché: Distributed Voice-and Web-Interfaced Market Information Systems under Rural Conditions. In: Proceedings of the 24th international conference on Advanced Information Systems Engineering (CAiSE). pp. 518–532. Springer (2012)
2. de Boer, V., Gyan, N.B., Bon, A., Tuyp, W., van Aart, C., Akkermans, H.: A Dialogue with Linked Data: Voice-based Access to Market Data in the Sahel. Semantic Web (2013)
3. Charlaganov, M., Cudré-Mauroux, P., Dinu, C., Guéret, C., Grund, M., Macicas, T.: The Entity Registry System: Implementing 5-Star Linked Data Without the Web. arXiv preprint arXiv:1308.3357 (2013)
4. Fernández, J.D., Martínez-Prieto, M.A., Gutiérrez, C., Polleres, A., Arias, M.: Binary RDF Representation for Publication and Exchange (HDT). Web Semantics: Science, Services and Agents on the World Wide Web 19, 22–41 (2013)
5. Gray, J., Chong, W., Barclay, T., Szalay, A., Vandenberg, J.: TeraScale SneakerNet: Using Inexpensive Disks for Backup, Archiving, and Data Exchange. arXiv preprint cs/0208011 (2002)
6. Guéret, C., Schlobach, S., De Boer, V., Bon, A., Akkermans, H.: Is data sharing the privilege of a few? Bringing Linked Data to those without the Web. ISWC2011 Outrageous ideas Track, Best Paper award pp. 1–4 (2011)
7. Gyan, N.B.: The Web, Speech Technologies and Rural Development in West Africa: An ICT4D Approach. Ph.D. thesis, Vrije Universiteit Amsterdam (2016)
8. Heeks, R.: ICT4D 2.0: The Next Phase of Applying ICT for International Development. Computer 41(6), 26–33 (2008)
9. Käbisch, S., Peintner, D., Anicic, D.: The Semantic Web. Latest Advances and New Domains: 12th European Semantic Web Conference, ESWC 2015, Portoroz, Slovenia, May 31 – June 4, 2015. Proceedings, chap. Standardized and Efficient RDF Encoding for Constrained Embedded Networks, pp. 437–452. Springer International Publishing, Cham (2015), `http://dx.doi.org/10.1007/978-3-319-18818-8_27`
10. Medhi, I., Ratan, A., Toyama, K.: Mobile-Banking Adoption and Usage by Low-Literate, Low-Income Users in the Developing World, pp. 485–494. Springer Berlin Heidelberg, Berlin, Heidelberg (2009), `http://dx.doi.org/10.1007/978-3-642-02767-3_54`
11. Pentland, A., Fletcher, R., Hasson, A.: DakNet: Rethinking Connectivity in Developing Nations. Computer 37(1), 78–83 (Jan 2004), `http://dx.doi.org/10.1109/MC.2004.1260729`
12. Rietveld, L., Beek, W., Schlobach, S.: LOD Lab: Experiments at LOD Scale. In: The Semantic Web-ISWC 2015, pp. 339–355. Springer (2015)
13. Verborgh, R., Vander Sande, M., Colpaert, P., Coppens, S., Mannens, E., Van de Walle, R.: Web-Scale Querying through Linked Data Fragments. In: LDOW (2014)