Business from technology

KSP: SPARQL-like knowledge sharing for resource restricted devices

M3 Semantic Interoperability Workshop

Jussi Kiljander

VTT Technical Research Centre of Finland

Introduction

- Fundamental idea in M3 is to utilize semantic technologies to achieve interoperability in pervasive computing environments.

- Typical devices (e.g. sensors, actuators, etc.) and communication technologies (e.g. 6LowPAN, BLE, etc.) utilized in pervasive computing are resource restricted.

- On the other hand, semantic technologies use verbose syntaxes that are not suitable for resource restricted devices.

- Additionally, the M3 communication protocol (i.e. SSAP) is based on XML format that both requires a large amount of memory and is slow to process in low capacity computing platforms.

Introduction cont.

- In M3 systems this problem has been typically solved by utilizing gateways which transform proprietary format data from low capacity devices to semantic format.

- This approach **reduces interoperability** and **complicates the system** as for each new device a new gateway is needed (*i.e.* gateways are application/device specific).

- We propose a novel **Knowledge Sharing Protocol (KSP)** that enables semantic communication in low capacity devices and networks by providing:
    - **SPARQL-like mechanisms** for accessing and manipulating knowledge in smart spaces **in a compact binary format**
    - methods for **simplifying application logic** and **reducing traffic** in low capacity networks.

Differences between KSP and SSAP

1) In KSP all operations are based on SPARQL 1.1.

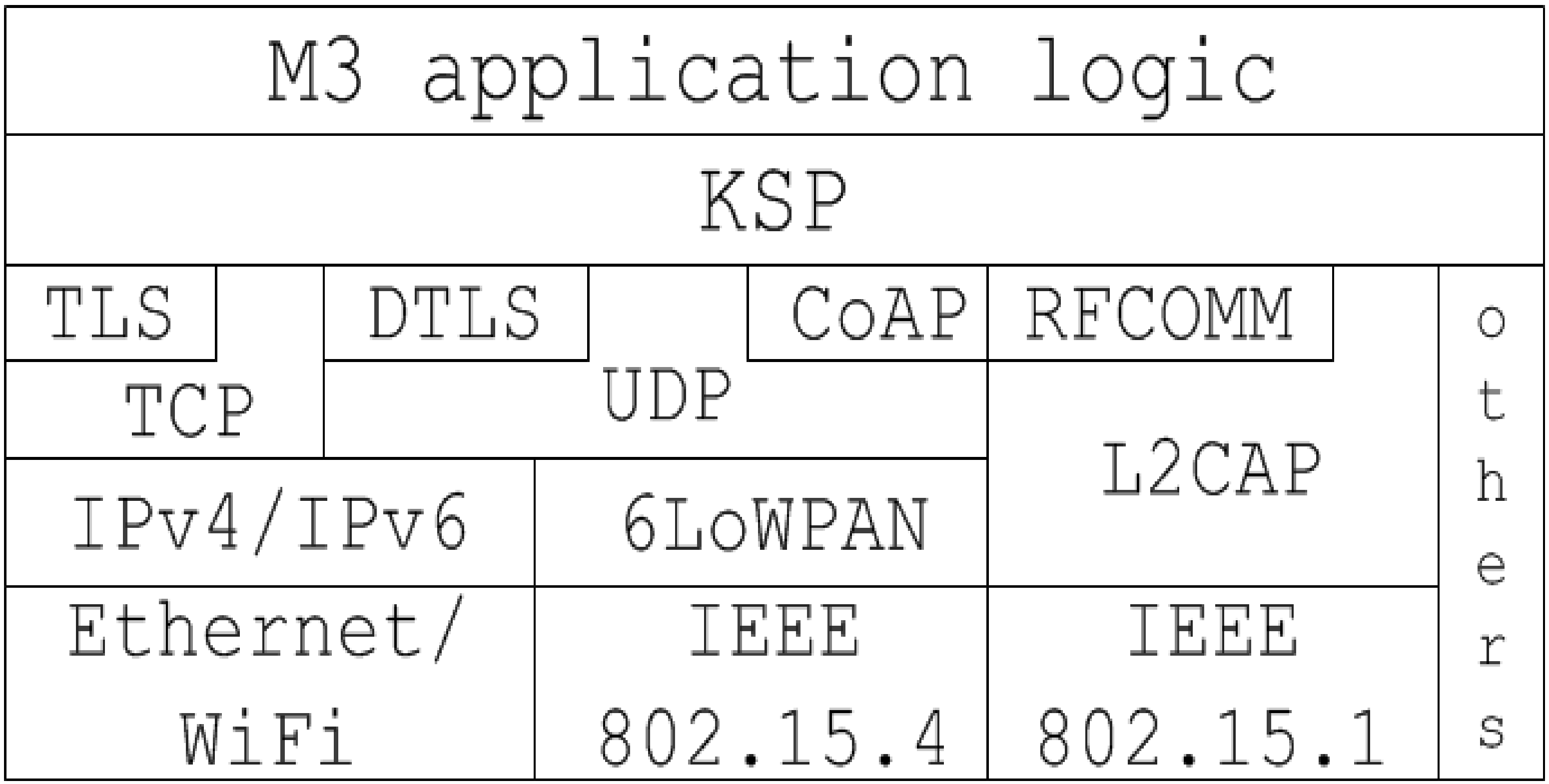
2) Binary format for the messages instead of XML

3) No *join* and *leave* operations

4) In KSP it is possible to define the maximum size for SIB responses

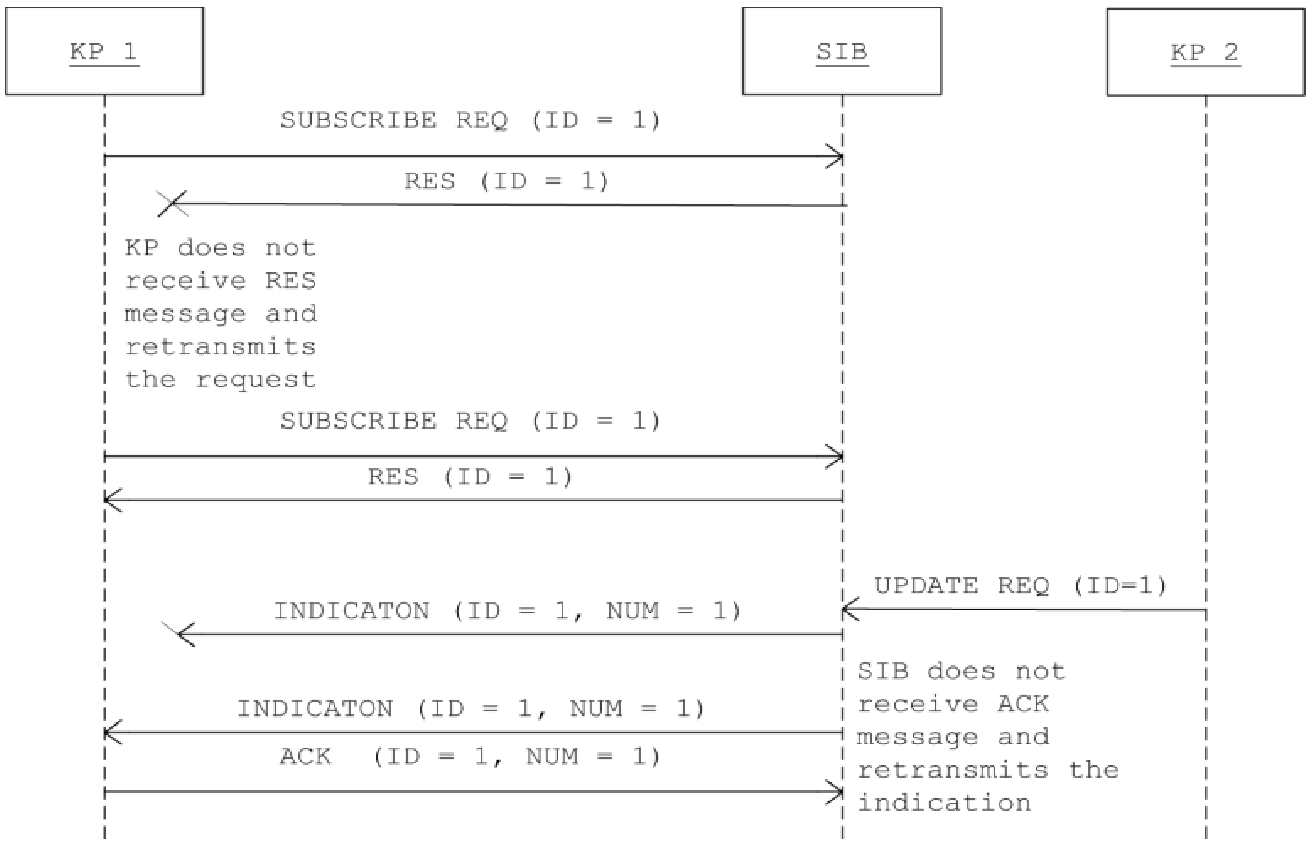
5) KSP defines persistent format also for update operations

Knowledge Sharing Protocol stack



Messaging Model

- Message types: Non-confirmable (NON), and Confirmable (CON), response (RES), indication (IND), **acknowledgement (ACK)**.



Message format: Header

- Messages consist of three parts: *header*, *data*, and *options*.
- Fixed size *header* field contains parameters such as *version*, *transaction type*, *request type*, and *transaction identifier* that are common for all transactions.
- The structure of the *header* field depends on the message type (REQ, RES, IND, or ACK) and the transport technology (e.g. TCP, UDP, BLE, etc.).

Header format for TCP

Request:

0 8 40 47 48 63

Version	Length	Transaction type	Request type	Transaction ID

Response/Indication:

0 8 40 47 48 63

Version	Length	Status code	Response type	Transaction ID

Header format for UDP

Request:

0	8	15	16	31
Version	Transaction type	Request type	Transaction ID	

Ack:

0	8	16	31
Version	0x00	Transaction ID	

Response:

0	8	15	16	31
Version	Status code	Response type	Transaction ID	

Indication:

0	8	15	16	32	48
Version	Status code	Response type	Transaction ID	Sequence number	

Transactions

Transaction type	Code
DELETE DATA	0x01
INSERT DATA	0x02
UPDATE DATA	0x03
DELETE	0x04
INSERT	0x05
UPDATE	0x06
SELECT	0x07
ASK	0x08
CONSTRUCT	0x09

Transaction type	Code
DELETE_PERSISTENT	0x0a
INSERT_PERSISTENT	0x0b
UPDATE_PERSISTENT	0x0c
SELECT_PERSISTENT	0x0d
ASK_ PERSISTENT	0x0e
CONSTRUCT_ PERSISTENT	0x0f
TERMINATE	0x10
RESET	0x11

Transaction type	Code
CREATE	0x12
DROP	0x13
COPY	0x14
MOVE	0x15

Message Format: Data Field

- In query and update messages the *header* field is followed by a *data* field which contains transaction specific information.

- In TERMINATE messages the *data* field is not needed.

- The structure of the *data* field depends on the operation type.
  - *E.g. SELECT, CONSTRUCT, ASK, DELETE, INSERT, etc.*

Data field format for query operations

SELECT REQ:

VC	Basic graph	OGC	[0-255]Optional graph

SELECT RES/IND:

PC	[0-255]URI	VC	TRC	RC	[0-65535]Result

[0-255] Variable

ASK REQ:

Basic graph	OGC	[0-255]Optional graph

ASK RES/IND:

Ask result

CONSTRUCT REQ:

Construct graph	Basic graph	OGC	[0-255]Optional graph

CONSTRUCT RES/IND:

PC	[0-255]URI	TRC	RC	[0-65535]Triple

Data field format for update operations

INSERT/DELETE DATA request:

Graph

UPDATE DATA request:

Delete graph	Insert graph

INSERT/DELETE request:

Graph	Basic graph	OGC	[0-255]Optional graph

UPDATE request:

Delete graph	Insert graph	Basic graph	OGC	[0-255]Optional graph

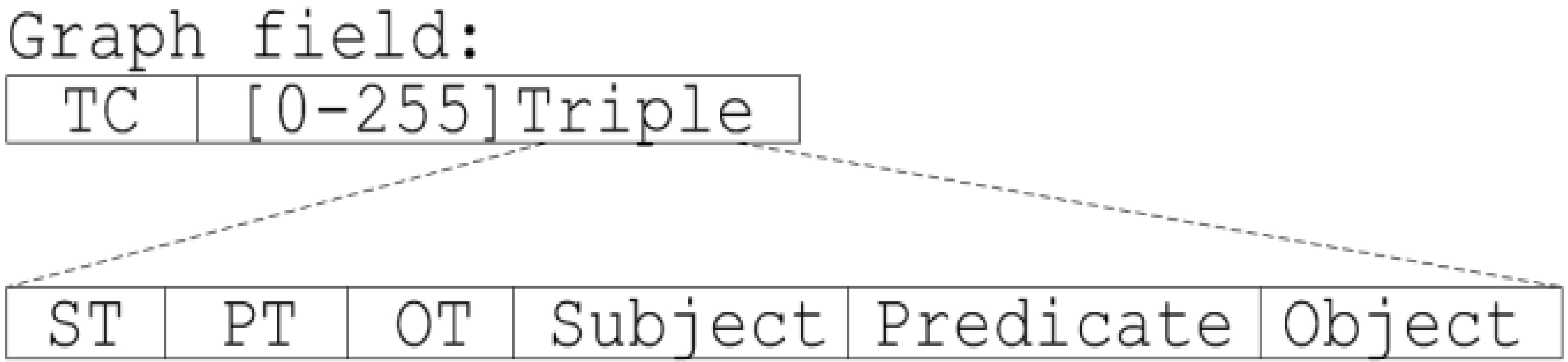
CREATE/DROP request:

GC	[0-255]Graph name

COPY/MOVE/ADD request:

Input graph name	Destination graph name

Encoding Format for RDF graph



- The *Graph* field consists of 8-bit triple count (*TC)* field and a zero or more (maximum 255) *Triple* fields.

- Each *Triple* field starts with 3-bit *ST*, 2-bit *PT*, and 3-bit *OT* fields, which specify the content of the following *Subject*, *Predicate*, and *Object* fields, respectively.

Type	Code
Empty	0x00
URI	0x01
Reserved Word	0x02
Variable	0x03
Literal	0x04

Field structure for RDF terms, variables and reserved words

URI field:

Prefix index	URI lenght	URI string

Literal field:

Literal type	Content

Variable field:

Variable index

Reserved word field:

Code

Type	Value
xsd:string	0x00
xsd:interger	0x01
xsd:float	0x02
xsd:dateTime	0x03
xsd:Boolean	0x04

Reserved Words

Word	Value
rdf:type	0x00
rdfs:Class	0x01
rdfs:subClassOf	0x02
rdfs:property	0x03
rdfs:subPropertyOf	0x04
rdfs:range	0x05
rdfs:domain	0x06
owl:TransitiveProperty	0x07
owl:SameAs	0x08
xsd:string	0x09
xsd:interger	0x0a
xsd:float	0x0b
xsd:dateTime	0x0c
xsd:Boolean	0x0d

Message Format and Semantics: Options field

- One of the main advantages of XML based protocols is extendibility.
- In KSP options are a way to achieve a certain level of extendibility in a non-XML protocol.
- Options also enable to create more compact messages by leaving out parts that are not needed in the particular message.

Option	Code
PREFIX	0x00
DELETE GRAPH	0x01
INSERT GRAPH	0x02
QUERY GRAPH	0x03
FILTER	0x04
SOLUTION MODIFIER	0x05
BIND	0x06
MAX RESPONSE SIZE	0x07
CREDENTIALS	0x08

Options field: encoding

Prefix field:

PC	[0-255] URI

Delete/Insert/Query graph field:

GC	[0-255] Graph name

Filters field:

FC	[0-255] Filter

TC	[0-255]Token

Type	(Operand)

Binds field:

BC	[0-255] Bind

TC	[0-255]Token	Variable index

Type	(Operand)

Credentials field:

KP ID	Password

Solution modifier field:

Order	Limit flag	Offset flag	Distinct
Unused	(Variable index)	(Limit)	(Offset)

Max response size field:

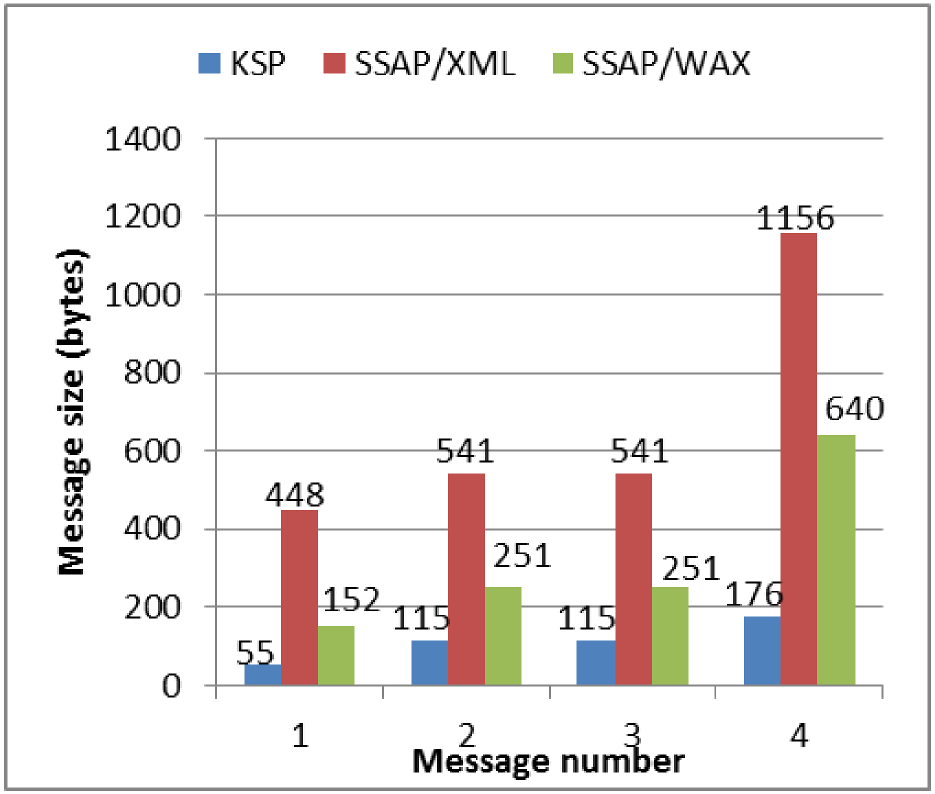
Size

Evaluation

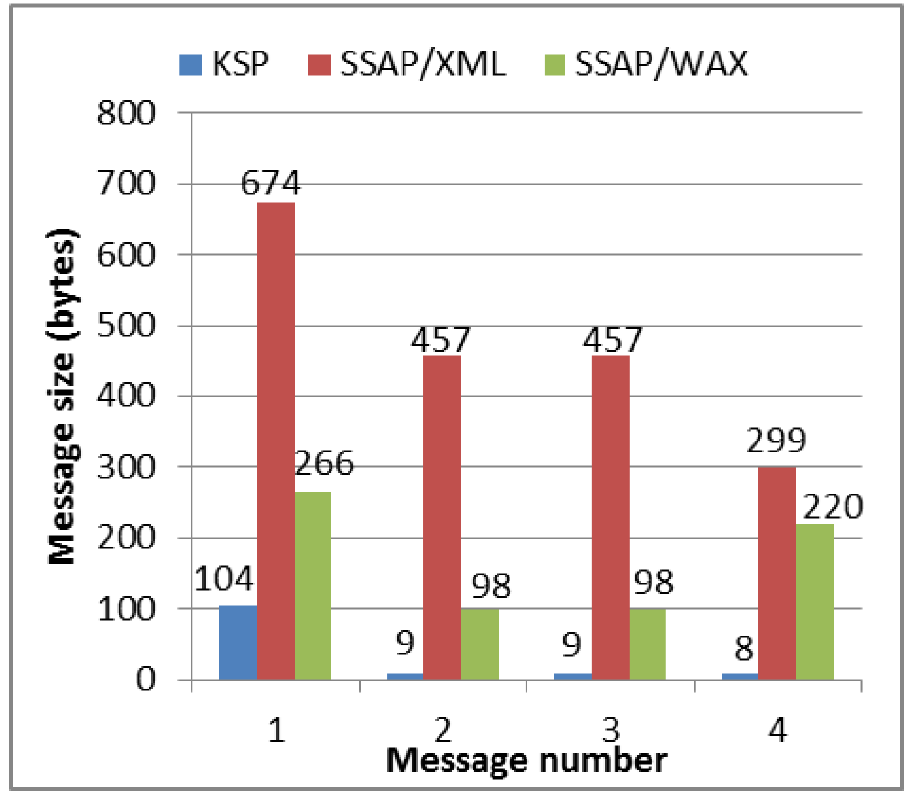
- Comparison of message sizes of KSP, SSAP/XML, and SSAP/WAX protocols in Smart Greenhouse demonstration.
- The KSP messages were on average 87.08% and 70.09% shorter than the SSAP/XML and SSAP/WAX messages respectfully.
- We also demonstrated how a KP implementation can be significantly simplified with persistent update operations.

Evaluation

requests



responses



Drawbacks and Limitations

- Binary format limits both the amount and the size of entities such as *prefixes*, *graphs*, *triples* and *results*.

- KSP requires a good application programming interface (API) because the binary format is not suitable to be used by developers as such.

- Some of the SPARQL 1.1 functionalities are not supported by the KSP because they would have made the KSP too complicated.
  - E.g. the current version of the KSP does not support DESCRIBE queries, Property paths, Aggregates and Subqueries, and many SPARQL functions.

Conclusions and Ideas for Future work

- By providing more **compact messages** and operations that **simplify the application logic** the KSP is more suitable for low capacity devices and networks than the official SSAP.

- The KSP has certain **limitations** compared to standard SPARQL 1.1 that might restrict its use in certain situations.

- The KSP can be extended, but there is **also a need for M3 protocol that supports the official SPARQL 1.1** as such.

- In the future, we should decide **what do with the SSAP**.
  - replace with HTTP/SPARQL or CoAP/SPARQL?
  - design SSAP v2.0?

Thank You!