RDF Provisioning for the Internet of Things

Henning Hasemann, Alexander Kröller, Max Pagel
TU Braunschweig
RDF Provisioning for the Internet of Things

Henning Hasemann, Alexander Kröller, Max Pagel
TU Braunschweig
The world as it should be
The world as it should be

sensor
The world as it should be

sensor

Application / Appliance
The world as it should be

sensor

6LowPAN, CoAP

access point, gateway

IPv6, HTTP, REST

Application / Appliance
The world as it should be

Sensor

6LoWPAN, CoAP

RDF (e.g., W3C SSN-XG)

access point, gateway

IPv6, HTTP, REST

Application / Appliance
The world as it should be

- sensor
- 6LoWPAN, CoAP
- access point, gateway
- IPv6, HTTP, REST
- RDF (e.g., W3C SSN-XG)

Application / Appliance

Linked Data Cloud
This talk: RDF on the sensor
**RDF** represents facts as *subject–predicate–object* triples:

```
sensor01 hasValue 22.7
```

**W3C SSN-XG** ontology for sensors (SensorML superset)

Plus: Data can be linked to arbitrary other datasets. Works with closed data too!

**Goal:** Sensor “is” a self-rewriting RDF document:

```xml
<rdf:Description rdf:about="http://spitfire.ibr.cs.tu-bs.de/be-0001/b4ec27c5-d543-496a-b2bf-a960134dcb37/2/sensor#sensor">
  <j.2:hasValue rdf:datatype="http://www.w3.org/2001/XMLSchema#double">22.7</j.2:hasValue>
  <j.2:hasLocation rdf:resource="http://sws.geonames.org/2945024"/>
  <j.2:hasLocation rdf:resource="http://rooms-bs.sytes.net/static/descriptions#Room4"/>
  <j.0:date>10.10.2012 20:15:00</j.0:date>
  <j.4:hasMeasurementCapability rdf:resource="http://spitfire.ibr.cs.tu-bs.de/be-0001/b4ec27c5-d543-496a-b2bf-a960134dcb37/2/sensor#measurementCapability"/>
  <j.4:feature0fInterest rdf:resource="http://rooms-bs.sytes.net/static/descriptions#Room4"/>
</rdf:Description>
```
Problem Space

Typical sensor (Jennic-based iSense):
• Specialized firmware (or Contiki, TinyOS etc)
• IEEE 802.15.4 radio (no direct IP connectivity)
• ~100kB of RAM shared for code, heap, data
• No external flash. No filesystem.
Problem Space

Typical sensor (Jennic-based iSense):
• Specialized firmware (or Contiki, TinyOS etc)
• IEEE 802.15.4 radio (no direct IP connectivity)
• ~100kB of RAM shared for code, heap, data
• No external flash. No filesystem.

→ Solution needs to be
Problem Space

Typical sensor (Jennic-based iSense):
• Specialized firmware (or Contiki, TinyOS etc)
• IEEE 802.15.4 radio (no direct IP connectivity)
• ~100kB of RAM shared for code, heap, data
• No external flash. No filesystem.

→ Solution needs to be
• **efficient** (code size)
Problem Space

Typical sensor (Jennic-based iSense):
• Specialized firmware (or Contiki, TinyOS etc)
• IEEE 802.15.4 radio (no direct IP connectivity)
• ~100kB of RAM shared for code, heap, data
• No external flash. No filesystem.

→ Solution needs to be
• **efficient** (code size)
• **efficient** (communications)
Problem Space

Typical sensor (Jennic-based iSense):
• Specialized firmware (or Contiki, TinyOS etc)
• IEEE 802.15.4 radio (no direct IP connectivity)
• ~100kB of RAM shared for code, heap, data
• No external flash. No filesystem.

→ Solution needs to be
  • efficient (code size)
  • efficient (communications)
  • efficient (data storage)
Problem Space

Typical sensor (Jennic-based iSense): :
- Specialized firmware (or Contiki, TinyOS etc)
- IEEE 802.15.4 radio (no direct IP connectivity)
- \( \sim 100\text{kB} \) of RAM shared for code, heap, data
- No external flash. No filesystem.

\[ \rightarrow \text{Solution needs to be} \]
- **efficient** (code size)
- **efficient** (communications)
- **efficient** (data storage)
- **platform-independent**
The Wiselib RDF Provider

- Sensor Data Provider
- Serializer
- RDF Service Broker
- CoAP
- B2B
- Client

Notifies:
- RDF Sensor Data
- RDF Documents
- RDF Providers

Wiselib TupleStore
The Wiselib RDF Provider

Based on the Wiselib: compiles for 10+ platforms, efficient C++
The Wiselib RDF Provider

Based on the Wiselib: compiles for 10+ platforms, efficient C++

Wiselib TupleStore
- Stores RDF triples
- Multiple documents
- Optional compression
- Configurable code vs. RAM vs. energy trade-offs

Sensor Data Provider
- RDF Sensor Data

RDF Documents
- RDF Sensor Data

Serializer
- notifies

Wiselib TupleStore
- Stores RDF triples
- Multiple documents
- Optional compression
- Configurable code vs. RAM vs. energy trade-offs

CoAP
B2B
…

Client
- RDF Service

CoAP
B2B
…

Based on the Wiselib: compiles for 10+ platforms, efficient C++
The Wiselib RDF Provider

**Based on the Wiselib:**
compiles for 10+ platforms, efficient C++

**Sensor Data Provider**
- Updates sensor data in the TS

**Wiselib TupleStore**
- Stores RDF triples
- Multiple documents
- Optional compression
- Configurable code vs. RAM vs. energy trade-offs

**Components:**
- Sensor Data Provider
- RDF Sensor Data
- Serializer
- RDF Documents
- RDF Service
- CoAP
- B2B
- ...
The Wiselib RDF Provider

Based on the Wiselib: compiles for 10+ platforms, efficient C++

Sensor Data Provider
- Updates sensor data in the TS

Wiselib TupleStore
- Stores RDF triples
- Multiple documents
- Optional compression
- Configurable code vs. RAM vs. energy trade-offs

Service Broker
- Provides document-level access
- Manages notifications

Wiselib TupleStore
- Stores RDF triples
- Multiple documents
- Optional compression
- Configurable code vs. RAM vs. energy trade-offs

Client

Serializer

RDF Documents

RDF Service

CoAP
B2B

...
Represent device with multiple RDF documents (TupleStore handles repetition)
Represent device with multiple RDF documents (TupleStore handles repetition)

Documents per device:

- NODE (device description)
- .well-known (list of documents, services)
Represent device with multiple RDF documents (TupleStore handles repetition)

Documents per device:
- **NODE** (device description)
- **.well-known** (list of documents, services)

Documents per sensor:
- **MINIMAL** (just the reading)
- **INTRINSIC** ( + basic info)
- **COMPLETE** (all available info)
Broker Protocols I: B2B Command Interface

- Simplest access: Command Interface, tailored for broker-to-broker communication
- Addresses problem: Tuples usually > 1 MTU

- Element-wise transmission + Command (insert/delete), with transaction logic

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
<th>Operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x2304</td>
<td>0xac48</td>
<td></td>
<td>INSERT</td>
</tr>
<tr>
<td>0xab80</td>
<td>0x16c0</td>
<td>0x30b4</td>
<td></td>
</tr>
</tbody>
</table>

... transaction table

<table>
<thead>
<tr>
<th>TS Dictionary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x2304<a href="http://www.ibr.cs.de..">http://www.ibr.cs.de..</a></td>
</tr>
<tr>
<td>0xac48<a href="http://www.w3.org/...">http://www.w3.org/...</a></td>
</tr>
<tr>
<td>0x1074„Wiselib RDF Provider“</td>
</tr>
</tbody>
</table>

...
Document-Level Interface

- operates on RDF documents
- usually CoAP service requests from Internet

Operations:
- GET \[documentID]\]
- POST \[documentIDs\] \[tuples]\]
- DELETE \[documentID\] \[tuple]\]
- SUBSCRIBE \[documentID\] \[callback]\]
- UNSUBSCRIBE \[subscriptionID]\]

Can use different serializations, for now:
Google ProtoBuf and **SHDT**... (RDF-XML too heavy?)
Streaming HDT

**HDT** [Fernandez et al. ’10]

<table>
<thead>
<tr>
<th>Header</th>
<th>(General meta information)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td><a href="http://www.tu-bs.de/kroeller/">http://www.tu-bs.de/kroeller/</a></td>
</tr>
<tr>
<td>1</td>
<td><a href="http://www.w3.org/1999/02/22-rdf...">http://www.w3.org/1999/02/22-rdf...</a></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>297</td>
<td><a href="http://www.w3.org/People/EM/...">http://www.w3.org/People/EM/...</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dictionary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>297</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Triples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subjects</td>
</tr>
<tr>
<td>33</td>
</tr>
<tr>
<td>75</td>
</tr>
<tr>
<td>53</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

“Header-Dictionary-Triples” (HDT)

- **Standard RDF serialization**
- **Highly efficient encoding**

**Drawback:** Requires full assembly of document before transmission

\[\hat{=} \] halving usable capacity
**HDT** [Fernandez et al. ’10]

**Header**
(General meta information)

**Dictionary**

<table>
<thead>
<tr>
<th>ID bitwidth</th>
<th>ID space size</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td><a href="http://www.tu-bs.de/kroeller/">http://www.tu-bs.de/kroeller/</a></td>
</tr>
<tr>
<td>1</td>
<td><a href="http://www.w3.org/1999/02/22-rdf/">http://www.w3.org/1999/02/22-rdf/</a></td>
</tr>
<tr>
<td>297</td>
<td><a href="http://www.w3.org/People/EM/">http://www.w3.org/People/EM/</a></td>
</tr>
</tbody>
</table>

**Triples**

<table>
<thead>
<tr>
<th>Subjects</th>
<th>Predicates</th>
<th>Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>33</td>
<td>72</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>29</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td>53</td>
<td>91</td>
</tr>
<tr>
<td></td>
<td>75</td>
<td>11</td>
</tr>
</tbody>
</table>

**“Streaming” HDT (SHDT)**

**Header**
(General meta information)

**Stream**

<table>
<thead>
<tr>
<th>Subjects</th>
<th>Predicates</th>
<th>Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>* 121</td>
<td><a href="http://www.tu-bs.de/kroeller/">http://www.tu-bs.de/kroeller/</a></td>
<td></td>
</tr>
<tr>
<td>* 23</td>
<td><a href="http://xmlns.com/foaf/0.1/name">http://xmlns.com/foaf/0.1/name</a></td>
<td></td>
</tr>
<tr>
<td>* 45</td>
<td>„Alexander Kröller“</td>
<td></td>
</tr>
<tr>
<td>121</td>
<td>23</td>
<td>45</td>
</tr>
<tr>
<td>* 78</td>
<td><a href="http://xmlns.com/foaf/0.1/title">http://xmlns.com/foaf/0.1/title</a></td>
<td></td>
</tr>
<tr>
<td>* 23</td>
<td>„Mr.“</td>
<td></td>
</tr>
<tr>
<td>121</td>
<td>78</td>
<td>23</td>
</tr>
</tbody>
</table>

...
Streaming HDT

- Document usually transmitted from IoT device (constrained) to Internet host (unconstrained)
- SHDT exploits asymmetry

- **Sender** has freedom to decide ID space, mix dictionary/triples, re-use IDs.
- Allows for encoding on-the-fly, needs buffer for 1 packet plus hash table (any size)
Evaluation – Streaming HDT

Serialization size


...
Evaluation — Code Size

![Bar chart comparing code size for iSense 5139 and iSense 5148]

- **iSense 5139**
  - TS & RDF Broker: 11.3k
  - CoAP: 11.1k
  - PB: 3.1k
  - SHDT: 1.8k

- **iSense 5148**
  - TS & RDF Broker: 6.0k
  - CoAP: 8.2k
  - PB: 1.6k
  - SHDT: 1.4k
  - B2B: 2.5k

The chart illustrates the code size for each component across both devices, with CoAP, PB, SHDT, and B2B being key factors in the overall size.
Evaluation — Code Size

<table>
<thead>
<tr>
<th>Platform</th>
<th>Contiki MicaZ</th>
<th>TinyOS MicaZ</th>
<th>TinyOS TelosB</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoAP</td>
<td>9.6k</td>
<td>9.0k</td>
<td>7.6k</td>
</tr>
<tr>
<td>B2B</td>
<td>4.0k</td>
<td>3.7k</td>
<td>2.2k</td>
</tr>
<tr>
<td>SHDT</td>
<td>1.7k</td>
<td>1.6k</td>
<td>1.0k</td>
</tr>
<tr>
<td>TS &amp; RDF Broker</td>
<td>9.1k</td>
<td>9.2k</td>
<td>6.3k</td>
</tr>
<tr>
<td>PB</td>
<td>2.9k</td>
<td>1.5k</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>21.6k</td>
<td>20.4k</td>
<td>15.4k</td>
</tr>
<tr>
<td></td>
<td>21.1k</td>
<td>19.8k</td>
<td>14.9k</td>
</tr>
</tbody>
</table>

Hasemann, Kröller, Pagel – RDF Provisioning for the Internet of Things – 10/26/2012
Contributions & Summary

“Node as RDF document”

- Modular & platform-independent RDF Provider
- Configurable protocols
- Configurable serialization,
  + new SHDT format

Upcoming: External memory data structures (e.g., Flash, SD)
Contributions & Summary

“Node as RDF document”

- Modular & platform-independent RDF Provider
- Configurable protocols
- Configurable serialization, + new SHDT format

Upcoming: External memory data structures (e.g., Flash, SD)

The End. Thank You!