Physical Markup Language (PML)
EPC Information Services
Middleware
Physical Markup Language (PML)

• Why do we need PML and what is it?
• PML ‘Core’ for communicating sensor information
• Examples of sensors and markup
Physical Markup Language (PML) – why do we need it?

- Need a common language for intercommunication
Physical Markup Language (PML) – what does it do?

• Provides a collection of standardized vocabularies to represent and distribute EPC-related information

• Examples include
  – observations by sensors such as RFID reads
  – configurations e.g. for RFID readers
  – e-commerce documents

• PML is only an interchange and communication format
  – Makes no assumption about how data is stored
  – Is not a product description/markup language
PML – language design

- Re-use of existing standards wherever possible
  - For the language itself
  - For the information marked up – timestamps, identity, sensors
- Core uses W3C XML Schema language [XSD]
  - Based on RosettaNet XML design methodology
  - Well-defined constraints, rigid document structure and content
  - Can use existing tools for creating, validating and parsing
- Simple but expressive
- Open and Readable
  - markup tag-names & values are human-readable, not cryptic
  - independent of transport protocol and data storage format
  - does not require proprietary authoring/processing tools
Communicating sensed information – PML Core

- PML Core is one vocabulary under the PML umbrella
- It represents data captured by EPC Network sensors
  - Marks up physical properties being observed
  - Representation with little interpretation, e.g. Reader ID, Timestamp, Tag ID
- Allows for a variety of sensors and capabilities
  - Sensors include RFID and barcode readers, temperature sensors
  - Tags may have additional memory, on board processing etc
- Allows for a hierarchy of sensor observations
  - An RFID tag may have an on-board temperature sensor
Sensors
Devices which make measurements of physical properties and entities

Observables
Physical properties or entities

Observations
Measurements made by the sensor, including timestamp

PML Core sensor model
What is a sensor?

- Automatic Identification Sensors
  - Bar Code Scanner
  - RFID Reader

- Positioning Sensors
  - GPS

- Optical Sensors
  - Temperature

- Environmental Sensors
  - Humidity
PML Core example – epc tag read

- RFID reader detects a tag in its read range

```xml
<pmlcore: Sensor>
  <pmluid:ID>urn:epc:1:4.16.36</pmluid:ID>
  <pmlcore:Observation>
    <pmlcore:DateTime>2002-11-06T13:04:34-06:00</pmlcore:DateTime>
    <pmlcore:Tag>
      <pmluid:ID>urn:epc:1:2.24.400</pmluid:ID>
    </pmlcore:Tag>
  </pmlcore:Observation>
</pmlcore: Sensor>
```
PML Core example – tag with memory

Class 2 tag with tag memory

- RFID reader detects a Class 2 tag (with extra data storage) in its read range

```xml
<pmlcore:Sensor>
  <pmluid:ID>urn:epc:1:4.16.36</pmluid:ID>
  <pmlcore:Observation>
    <pmlcore:DateTime>2002-11-06T13:04:34-06:00</pmlcore:DateTime>
    <pmlcore:Tag>
      <pmluid:ID>urn:epc:1:2.24.400</pmluid:ID>
      <pmlcore:Data>
        <pmlcore:XML>
          <EEPROM xmlns="http://sensor.example.org/">
            <FamilyCode>12</FamilyCode>
            <ApplicationIdentifier>123</ApplicationIdentifier>
            <Block1>FFA0456F</Block1>
            <Block2>00000000</Block2>
          </EEPROM>
        </pmlcore:XML>
      </pmlcore:Data>
    </pmlcore:Tag>
  </pmlcore:Observation>
</pmlcore:Sensor>
```
PML Core example – tag with sensor

Class 4 tag with temperature sensor

• RFID reader detects Class 4 tag with a logging temp sensor in its read range

```xml
<pmlcore:Sensor>
  <pmluid:ID>urn:epc:1:4.16.36</pmluid:ID>
  <pmlcore:Observation>
    <pmlcore:DateTime>2002-11-06T13:04:34-06:00</pmlcore:DateTime>
    <pmlcore:Tag>
      <pmluid:ID>urn:epc:1:2.24.400</pmluid:ID>
      <pmlcore:Sensor>
        <pmluid:ID>urn:epc:1:12.8.128</pmluid:ID>
        <pmlcore:Observation>
          <pmlcore:DateTime>2002-11-06T11:00:00-06:00</pmlcore:DateTime>
          <pmlcore:Data>
            <pmlcore:XML>
              <TemperatureReading xmlns="http://sensor.example.org/">
                <Unit>Celsius</Unit>
                <Value>5.3</Value>
              </TemperatureReading>
            </pmlcore:XML>
          </pmlcore:Data>
        </pmlcore:Observation>
      </pmlcore:Sensor>
    </pmlcore:Tag>
  </pmlcore:Observation>
</pmlcore:Sensor>
```

.........

...............
Wired temperature sensor

- Wired temperature sensor logs ambient temperature periodically

```
<pmlcore:Sensor>
  <pmlcore:Observation>
    <pmlcore:DateTime>2002-11-06T13:04:34-06:00</pmlcore:DateTime>
    <pmlcore:Data>
      <pmlcore:XML>
        <TemperatureReading xmlns="http://sensor.example.org/">
          <Unit>Celsius</Unit>
          <Value>22.24</Value>
        </TemperatureReading>
      </pmlcore:XML>
    </pmlcore:Data>
  </pmlcore:Observation>
</pmlcore:Sensor>
```
EPC information service (EPCIS)
EPC information service (EPCIS)

• What data does EPCIS relate to?
• How do we get useful information from the EPCIS?
• Worked example
EPC information service – the ‘network database’

- EPC information service
EPC information service
– the ‘network database’

• Provides information about an EPC-tagged object
  – EPC is used as a database lookup key

• Really only provides the *interface* to this information
  – May interface to existing databases, apps & information systems
  – May provide its own persistent data storage

• Two-way data flow
  – Raw data sent to EPCIS for storage
  – Queries sent to EPCIS for information retrieval

• EPCIS previously known as the PML Server or PML Service
  – Won’t necessarily store or markup in PML
What data is accessible through EPCIS?

<table>
<thead>
<tr>
<th>Data Accessible through EPCIS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Timestamped historical data</strong></td>
</tr>
<tr>
<td>Observations (Tag readings)</td>
</tr>
<tr>
<td>Measurements (sensor data)</td>
</tr>
<tr>
<td>Symbolic Location/Containment</td>
</tr>
<tr>
<td>EPC &lt;-&gt; Transaction ID</td>
</tr>
</tbody>
</table>
How can the data be turned into useful information?

• What sort of questions may be asked?
  – “Where is the nearest red sweater in a size 4?”
  – “Where are the 5 CDs that were supposed to be in the last order?”
  – “This case of meat is tainted. Which items crossed paths with it?”

• Data required is available, but need to retrieve it!
  – Queries are complex
    1. May involve sophisticated data storage and processing
    2. May require data from multiple EPCIS’s and other applications
Sophisticated data storage and processing

• Need to store & retrieve attribute data quickly & efficiently
  – Existing systems do this, but may need extension

• Attribute data defined at different levels
  – Product type – e.g. type, colour
  – Batch or instance level – e.g. sell-by date, origin

• Need to express properties in an unambiguous way
  – Independent of language differences e.g. color/couleur/Farbe
  – Fully specified units e.g. metres, inches, feet

• Need to ‘fully qualify’ properties
  – e.g. gross weight vs. net weight
Sophisticated data storage and processing

- Need to store & retrieve historical data quickly & efficiently
  - Existing systems may not readily support this

- EPCIS receives data in triples:
  - EPC, reader ID, timestamp

- We might want to retrieve information in the form:
  - The readers that saw tag A
  - The tags that reader R saw
  - Reads that were detected from time t1 to time t2

- Involve reverse lookups of large data sets
  - One approach is to represent using a 3-D data space
  - Each triple of data is plotted as one point
Sophisticated data storage and processing

- Tell me all readers which saw tag e5
Sophisticated data storage and processing

- Tell me all tags seen by reader r9
Sophisticated data storage and processing

• Tell me all tags seen in the time range $t_1 - t_2$
Data from multiple EPCIS’s

• Need to break down complex queries
  – EPCIS will only handle queries which it can answer directly
complex query example

“Where is the nearest red sweater of size 4?”

1. Resolve any linguistic ambiguities
e.g. Does ‘nearest’ mean ‘shortest distance’ or ‘quickest time’?

2. Tell me all EPCs on local EPCIS subject to:
   - static attributes: Colour = red
     Size=4
   - dynamic attributes: In Stock = yes
     Already Sold = no

=> list of EPCs, \{e\}
complex query example

“Where is the nearest red sweater of size 4?”

3. For each of these EPCs, tell me which readers have seen it in the last hour?

   SELECT * FROM Reader_data WHERE tagEPC = e
   AND TIMESTAMP > (NOW – 3600)
   => list of readers, \{r\}

4. For each of these readers, tell me your current \((x,y,z)\) location static attributes: Cartesian Location
   (from location data)
   => \((x,y,z)\) for each reader \(r\)
complex query example

“Where is the nearest red sweater of size 4?”

5. For each of these \((x,y,z)\) co-ordinates, sort in order of increasing distance from my position at \((x_0, y_0, z_0)\)

\[
\text{ORDER BY } (x-x_0)^2 + (y-y_0)^2 + (z-z_0)^2 \text{ ASC}
\]

=> ordered list of readers, \(\{r\} – \text{nearest first}\)

6. For the nearest \(n\) readers \(r\), obtain human-readable location static attributes: Human-Readable Location (from location data)

=> “Aisle 5; shelf 3 from the top” etc.
Savant and middleware

- The problem
- Original concept
- Reference design specification
- Current SAG activity
- BIS integration
Basic problem

• When the vision becomes reality, there’s a lot of data
  – Item level detection for literally billions of items
  – Many detection locations
  – Unique identification
  – Licence-plate only

• To make this work we need to
  – Minimise data communications
  – Make communications as efficient as possible
  – Make the system robust to communications outages
Original concept

- Hierarchical approach with two elements
  1. Data is processed as close to leaves as possible
  2. Information is cached where possible
     - Naturally scalable and robust
- Framework for implementation, called ‘Savant’
  - sa·vant (sa-vänt ) n.
     1. A learned person; a scholar.
     2. An idiot savant (an intellectually disabled person who exhibits extraordinary ability in a highly specialised area).
Original concept

Distribution centre example:

The ‘Savant’

Internal Savant

Internal Savant

Edge Savant

Edge Savant

Edge Savant

Dock door 1

Dock door 2

Storage bays
Specification via reference design

- Centre decided to build reference design
  - Freely available proof-of-concept
  - Basis for Savant specification

- 3 modules
  - Event management system (EMS)
  - Real-time in-memory data structure (RIED)
  - Task management system (TMS)
Event management system

- Supports reader adapters
  - Like printer drivers – interface with different readers
- Collects EPC data from readers in real time
  - Common format for data, via adapters
- Supports ‘event’ filters and loggers
  - Turn data into information as soon as possible
  - Event-based rather than state-based
- Provides buffering (caching) to prevent blocking
Real-time in-memory event database

• Support for tens of thousands of transactions per second
• Efficient support of persistent information
  – Multiple read-only snapshots of outdated information
  – Support theft applications etc.
• Provide a standard API to access and manipulate data
• Limited support for sophisticated features
  – Complex queries
  – Limited query optimisation
  – No constraint maintenance
  – Efficient search and join
  – No trigger support
  – Limited DDL & DML
Task management system

• ‘Operating system’ to support EMS, RIED
  – Small memory footprint, multi-platform (incl. embedded systems)
  – VM with libraries loaded/updated on demand (redundant servers)
  – External interface for task scheduling
  – Persistent scheduler with ability to support restarts

• Supports
  – Remote operation, maintenance and task scheduling
  – Data gathering and communication (between savants, for e.g.)
  – Apps like shelf replenishment, theft and expiration alerts
“Savant is software that sits between tag readers and enterprise applications, providing a variety of computational functions on behalf of applications”

- Middleware to address unique requirements of EPC network
- Savant ‘hierarchy’ is much less applicable
  - One savant may perform a lot of processing
  - Existing middleware may provide higher level functionality
- May be considerable intelligence in readers
  - Sophisticated filtering, smoothing, reporting etc.
Extensibility and flexibility

• Emphasis on extensibility, not specific processing features
  – Specific EPC processing requirements vary between applications
  – As EPC matures, innovation and change in applications is likely
• Savant defined in terms of processing modules or services
  – Each provides a specific set of features
  – May be combined by the user to meet application needs
  – Modular ‘component’ structure, with Savant as ‘container’
• Processing modules talk to outside world via two interfaces
  – Reader interface
  – Application interface
Functionality supplied by processing modules

- **Standard processing modules**
  - Defined by EPCglobal specs
  - Must be provided by all Savant software
  - *Required* SPMs running in every deployed instance
  - *Optional* SPMs may be included or omitted by the user in a given deployed instance

- **User-defined processing modules**
Standard processing modules in v1.0

• **autoid.core**
  - Means for apps to learn what other processing modules are available and to get basic info about what readers are connected

• **autoid.readerproxy**
  - Means for applications to issue commands directly to readers

• In v1.0, most of functionality implemented with user-defined PMs;
  - Future versions may define additional standard PMs
  - ONS access, EPC IS access, counting, filtering, aggregation etc.
Additional interfaces

• Processing Modules may interact with
  – each other through APIs they define themselves
  – other external services via interfaces exposed by those services
  – A special case of this is one Savant interacting with another

• v1.0 spec doesn’t define how Processing Modules gain access to such external services
  – expected to change in the future
Application interface

• Provides standard connection to external applications
  – Existing enterprise “back end” applications
  – Possibly new EPC-specific applications or other Savants

• Protocol fully specified in terms of command sets
  – Each command set defined by a processing module

• Specified using a layered approach
  – One layer defines the commands and their abstract syntax
  – A lower layer specifies a particular syntax & protocol binding
  – Several bindings may be defined
Interfacing with existing BIS

• Savant designed to interface with BIS
  – Provide EPC data in suitable format/volume
  – Provide API for control of readers & low-level processing

• Combined with ONS and EPC IS is complete system
  – End users keen to promote open system approach
  – Enables some of the applications envisaged
  – BIS vendors traditionally keep things proprietary!
  – Ultimately BISs may be re-architected