OMA LWM2M in a Holistic Architecture for the Internet of Things

David Tracey
Dept. Of Computer Science,
University College Cork,
Cork, Ireland.

Cormac Sreenan
Dept. Of Computer Science,
University College Cork,
Cork, Ireland.

***Abstract*—Wireless Sensor Networks (WSNs) allow applications to interact with the physical world using nodes deployed in an Internet of Things (IoT). Application level protocols such as the Constrained Application Protocol (CoAP) and data models such as IPSO Smart Objects and the Open Mobile Alliance Lightweight Specification (OMA LWM2M) have the potential to provide greater application interoperability and to ease the difficulties imposed by the heterogeneous nature, limited development environments and interfaces of existing solutions. This paper describes an architecture using a tuple-space based library for the flow of data from sensors to applications with defined service abstractions. It also considers the OMA LWM2M Information Model in comparison to the DMTF Common Information Model. It presents a 'C' implementation of these models on our tuple-space running on the Contiki3.0 OS and considers the effectiveness of our architecture and its integration with the existing CoAP and OMA LWM2M implementations.**

***Index Terms*—Wireless Sensor Networks, IOT, Tuple Space, Data Model, OMA LWM2M**

I. Introduction

Definitions of IoT generally share the idea that IoT relates to the integration of the physical world with the Internet's virtual world[1]. IoT uses individually addressed, constrained devices in a distributed system, with sensing and active devices for physical phenomena. Although deployed in a variety of applications, such as environmental monitoring and healthcare, deployments are often dedicated/proprietary or specialized to optimise one particular aspect such as lifetime.

One aspect to be considered is how to store and represent the variety of data on constrained devices so that application software can understand data from sensors and actuators in the way people using browsers understand information on the Web[2]. The IP for Smart Objects (IPSO) alliance promotes the use of IP-based technologies, defined by standard organizations for smart objects in a range of interoperation use cases[1]. The IPSO Smart Object definitions are used in this paper and comprise mostly sensors/actuators. A broader range of smart objects is envisaged in [3], where in addition to sensing and logging, smart objects can act on their own and exchange information with humans. IPSO basic smart objects can be used to form composite objects. IPSO objects can be accessed with a URI and encapsulate sensor data, links and metadata. IPSO objects do not require the use of CoAP, but can be used to develop interoperable solutions with the Open Mobile Alliance Lightweight Machine to Machine Specification (OMA LWM2M)[4].

Our view is that flexible, service-based interoperable abstractions, with supporting data models such as OMA LWM2M, are key to moving IoT beyond isolated islands of sensor data to networks that are more easily deployed, developed and integrated with new services, particularly at the edges of the Internet. Hence, we think that IoT architectures should revolve around data. Our architecture[5] was designed to represent the node's local data and service roles for the nodes and applications, based on their role in the flow of data from sensor to application. This paper presents a summary of our architecture and how its data-centric approach and tuple based store (and templates) provide a set of abstractions to reduce the likelihood of isolated islands of data (due to proprietary/different standards).

This paper extends an existing implementation of the OMA LWM2M model on the Contiki3.0 OS to integrate with our architecture's novel design point of a tuple-based store for both local and remote node data with a simple API and defined service roles. This data-centric design allows code reuse and interoperability on constrained nodes and cloud services. This paper also considers the issues encountered in implementing the OMA LWM2M model.

The remainder of this paper is organised as follows. We present prior work in section II and an overview of our architecture in section III. Sections IV and V present and review a prototype integration of the OMA object model into our architecture. The paper concludes in section VI.

II. Existing WSN Model Approaches

WSN nodes, such as wismote[2] are constrained in terms of processing power, memory and energy consumption, making it a challenge to deliver the sensed data to application(s) and also support generic APIs and data models. This section considers some approaches which are independent of the wireless technology.

A. IPSO and OMA LWM2M

The REST architectural style represents Resources, e.g. a sensor, in specified formats, which are accessed by their Universal Resource Identifier (URI) using a defined set of verbs, such as GET, POST, PUT, DELETE in HTTP[6]. The Constrained Application Protocol (CoAP) is a RESTful protocol for constrained devices and networks, which provides resource discovery via the Resource Directory (RD) and an

[1] IP for Smart Objects (IPSO) Alliance, www.ipso-alliance.org

[2] www.wismote.org

"observe" flag in the CoAP GET Request to provide an observe/notify (publish/subscribe) model.

OMA LWM2M is an example of a RESTful approach using CoAP. In an IoT context it swaps "server" and "client" roles in that a node runs at least a CoAP Server and LWM2M Client, rather than being simply a client. LWM2M provides a simple and reusable object model with a set of interfaces for managing constrained devices, covering bootstrap, registration, information reporting, device management and service enablement. Figure 1 shows IPSO Smart Objects, LWM2M, CoAP and 6LOWPAN combine to give a uniform API and Data Model stack to provide end-to-end interoperability between constrained devices and services.

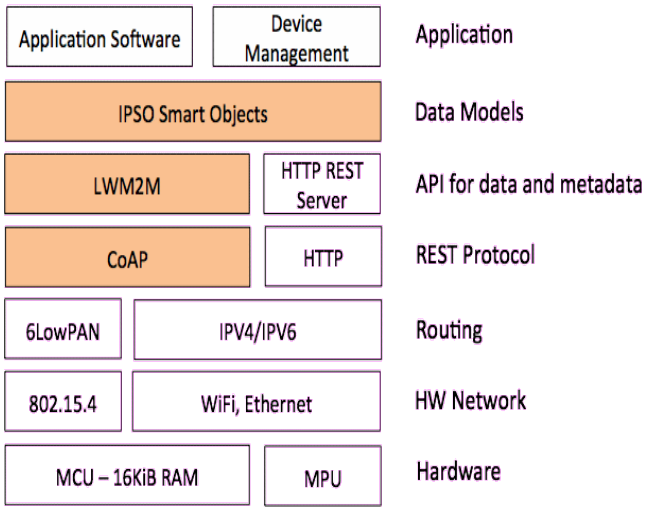

Fig. 1. OMA Protocol Stack[3]

IPSO Smart Objects cover a range of entities, including basic sensors and actuators. These basic objects are represented using a simple common data model and resource template in LWM2M. The model consists of Resources with a URI of object/instance/resource to identify a resource, e.g. 3303/0/5700, represents a "Sensor Value" (resource id 5700) in a "Temperature Sensor" (object ID 3303) instance (id of 0). More complex objects can be composed to represent items with multiple resources, e.g. an IPSO Thermostat(8300) may have IPSO temperature sensors, (3303), IPSO Setpoint (3308) and IPSO Actuation(3306) [7]. The implementation of CoAP on Contiki in [8] is used in the LWM2M and IPSO implementation[4] OMA LWM2M-supported devices are not yet widely available, so [9] uses a new LWM2M gateway between an LWM2M server and legacy devices that also integrates into the ETSI M2M architecture. A new client engine implementation for LWM2M on Contiki-based nodes is shown in [10].

B. Tuple Based Approaches

TeenyLIME [11] is a high level approach, built on TinyOS, which is based on a shared memory space (tuple space), derived from Linda[12]. TeenyLIME's deployment in a real-world application showed the usefulness of a tuple space approach in WSNs[13]. This approach allows different processes to use a limited number of simple operations to insert, read, and withdraw tuples from a tuple space and to provide asynchronous notifications for data of interest being added to the shared tuple space. LighTS[14], part of the LIME environment, provides a reduced tuple space holding context (location) information using the same primitives. LIME extended the local node tuple space into a federated tuple space into which tuples can be added, removed, but only when the nodes are in range of each other[15]. LIME is implemented in Java, limiting its applicability to more capable nodes, whereas TeenyLIME can run on constrained devices.

C. Other Approaches

TinyDB [16] considers the WSN as a distributed database, with a table where each column represents a sensor reading or node data and a SQL like query language (extended for periodic requests) with nodes supporting aggregation of data. While powerful, this approach can be considered limited by its table based approach and relational queries, especially in terms of handling events. A data-centric approach such as directed diffusion may be more suitable in certain cases, such as a request for information from a group of nodes or any node in a particular region, rather than the "normal" model of a request being made to a particular node[16]. It uses a publish and subscribe model where a node expresses an interest in data items using a set of attribute-value pairs. Each node keeps an interest cache with entries for each interest and nodes which can provide the relevant data will reply. Directed Diffusion is, however, tightly coupled to a query on demand data model where applications can accept aggregated data. An approach in [3] defines metamodels at different levels of abstraction considering functional and data perspectives to assist the analysis, design and implementation of smart objects.

III. Our Holistic Architectural Approach

OMA LWM2M provides solutions for end-to-end interoperability across networks and devices, but that it provides limited higher-level service abstractions beyond client/server. The objective of our architecture is to enable a wider deployment of WSNs while also providing consistent abstractions to enable the easier development of generic and more powerful applications to take advantage of sensor data. The key principle underlying our approach is that all WSNs are primarily about delivering sensed data to one or more applications (periodically, on-demand or asynchronously) or commands to actuators from applications. The approach is termed as holistic in that it considers the entirety of the flow of data between sensor and service(s), supported by lower layers, rather than being driven by each layer specifying its own behaviour in isolation.

A. HPP Architecture

This system model is supported by our Holistic Peer to Peer (HPP) Architecture[5]. This architecture includes a data model (dm) service layer, an object space layer and a local instrumentation layer. The data model service layer represents nodes and services (on node or Cloud) and holds its data in the object space layer. The object space layer is a data store, modeled as a tuple space, with a simple API and leases on

[3] http://openmobilealliance.org/wp-content/uploads/2015/02/data-models-2.gif
[4] https://github.com/contiki-os/contiki/tree/master/apps/oma-lwm2m

stored objects. The local instrumentation layer hides the platform specific sensor hardware implementations and also uses the object space layer to hold the data for the local node. Figure 2 illustrates the relationship of these layers with the object space holding objects from the local node and remote nodes. This separation of remote and local data allows data to be transferred or stored for forwarding to another node.

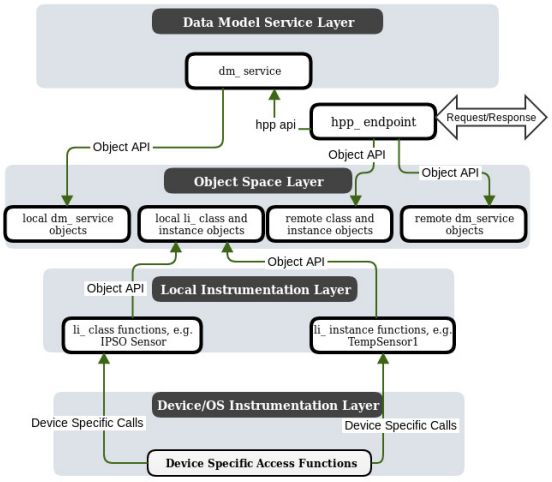

Fig. 2. Node Data Architecture

1) The Data Model and Service Abstractions

The DM Service layer is independent of a particular data model and provides a simple and flexible API for a data store To insulate the developer from network and node specifics, such as hardware and different node functionalities, the data model service layer uses defined roles for services and nodes, based on capabilities.The defined roles are:

1. DM_SINK_SRV - adds interest objects to its peers for data it wants to receive
2. DM_SOURCE_SRV – sends it sensor data to peers
3. DM_FORWARDER_SRV - passes messages to peers
4. DM_STORE_SRV - provides intermediate storage for data from remote peers, such as historic data
5. DM_AGGREGATOR_SRV- aggregates peer data
6. DM_MATCHER_SRV - provides advanced query matching

A node may play several roles according to its resources and a constrained node may only act as a DM_SOURCE_SRV, not even storing its own data or forwarding that of others. More capable nodes may cache or aggregate data.

2) The Object Space and Library

The object library provides resource constrained devices with a simple shared object space and associated API, using concepts from Linda's tuple space[11]. The object space is non-prescriptive about how it holds classes and instances, except that it requires the use of a template to hold the type of each object attribute and its methods. n object structure represents an object held in the object space, using the object's class template. Each object has a lease, allowing for the space to remove objects if leases are not renewed. A node adds the template defining the information model and the names of the properties it supports, i.e. to specify which properties of an object are instrumented. The template and instance are kept separately to allow for objects that represent a class. For resource constrained devices it also allows the template (or a reference) to be sent once to another node prior to the encoded instance. The actual definition of the template is transparent to the object space, although the current implementation uses a key-value pair based definition.

Furthermore, our object space can act as a cache for local and remote node data. The successful use of a cache enables reduced communication and so extends the battery life of WSN nodes. We have proposed the CacheL algorithm for WSN nodes [18], which uses an intrinsic lease associated with cache data in its cache replacement policy.

3) Local Instrumentation Layer

This layer hides the platform specific sensor hardware implementations. It provides get()/set() functions and method prototypes to access local node data and functionality. This aligns with the hardware/vendor specific implementations on nodes to access particular readings, e.g. a call to read a value from a register. The local instrumentation (li) layer treats each object attribute/property individually. Figure 3 shows these individual li_class_property properties are stored only once in a li_class_list and how the values of each li_instance are stored in a separate li_value_list. It also shows how it separates key and non-key properties, for information models that use keys to identify object instances (or table rows). It also allows resource constrained devices to allocate and set keys when the class is created, whereas non-key data in an instance changes and may be read by a dynamic getter function.

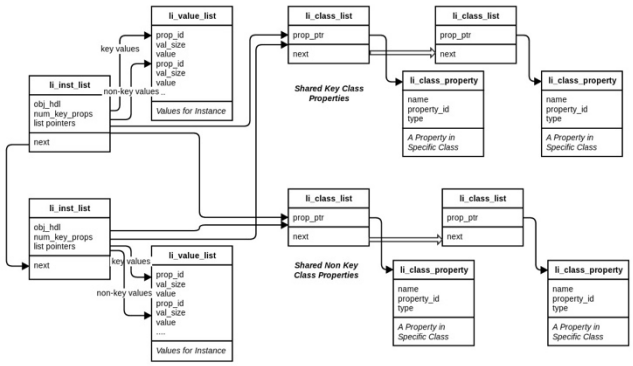

Fig. 3. Local Instrumentation Structures

This provides the flexibility required to map a rich data model to a resource constrained WSN device, as higher-level data models can be built up using a local instrumentation structure per attribute giving per attribute mapping to the underlying node functions or data. This also allows only those attributes supported by the node to be implemented, rather than having to store an object's unsupported attributes. This contrasts with how objects are normally inherited with all attributes, even if not required. We showed previously that it is straightforward to map this per property approach to a complex object such as used in CIM[5]. This approach is also very much

[5]http://www.dmtf.org/standards/cim

in line with the per property (or Resource in IPSO terms) approach used in OMA LWM2M.

4) The Holistic P2P Protocol (HPP)

The implementation in this paper integrates the layers from our architecture with CoAP for use with OMA LWM2M. IPSO objects can also be supported over our HPP protocol.

The HPP protocol is sufficiently simple for low capability devices to exchange sensor information independent of the underlying technology, while providing the resilience of a P2P protocol, together with leases, to handle intermittent connectivity. Figure 4. shows an example interaction, where a DM_SOURCE_SRV adds its service and node classes and instances to a DM_STORE_SRV on a node able to cache data. A DM_SINK_SRV queries this DM_STORE_SRV for its capabilities and then its node data, which may be returned from the DM_SOURCE_SRV or an intermediate DM_STORE_SRV (if cached there).

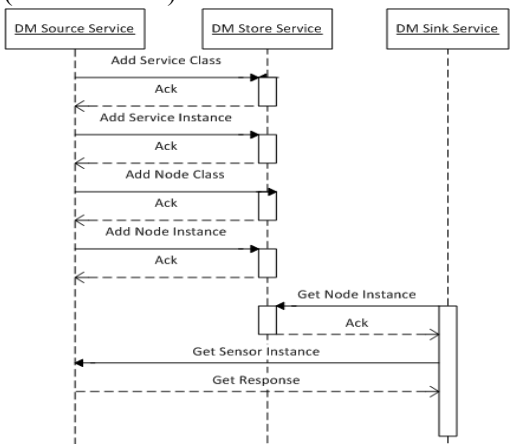

Fig. 4. Sample Service Interaction

IV. IMPLEMENTATION

We implemented the HPP data model elements comprising the data model service layer, local instrumentation layer and object space on Contiki3.0 and integrated them with the erbium and CoAP implementations (er-rest-example) in [8]. In order to demonstrate the flexibility of our architecture, we extended the implementation to OMA LWM2M using the IPSO and OMA applications in Contiki3.0. Note that this hpp code is not gateway or client specific.

A. Data Model Service

The data model service uses key-value pairs to store or send objects from/to a remote node and a local instrumentation form to encapsulate the node functions to access sensor data. The dm_service library helper functions, e.g. *dm_add_class()* and *dm_add_instance(),* handle both forms of objects.

B. Local Instrumentation Adapter

Figure 3 shows how locally instrumented data is implemented. A node allocates and sets up an li_class structure for each class to be stored locally. The li_class consists of a list of class_property_t, one per property of the class. It then sets up an li_instance structure for each object instance, e.g. a sensor. Each li_instance_property is linked to its single li_class_property definition which reduces memory use compared to having this in every instance. The li_class_property structure makes no assumptions about the object it is to be put in (it could be in several), giving the modelling flexibility outlined above. OMA LWM2M Contiki Implementation

The existing implementation in Contiki3.0 uses a set of structures to represent objects and resources, with enumerations for types and a context for parsing a request. The LWM2M object structure contains pointers to instances, which in turn contains pointers resources as below:

```
typedef struct lwm2m_object {
    uint16_t id;
    uint16_t count;
    const char *path;
    resource_t *CoAP_resource;
    lwm2m_instance_t *instances;
} lwm2m_object
```

The ipso-example code initialises the LWM2M engine in a thread that calls *lwm2m_engine_init(),* followed by *lwm2m_engine_register_default_objects()* to set up a device object for the node, then *ipso_objects_init()* to initialise the supported objects, e.g. *ipso_temperature_init()*. It then loops processing events. These object init() methods have code like the following for the temperature resource, showing the Object Id, type and value as per above lwm2m_resource. The callback will be a method to access the real values.

```
LWM2M_RESOURCES(temperature_resources,
   LWM2M_RESOURCE_CALLBACK(5700,
                      {temp, NULL, NULL}),
   LWM2M_RESOURCE_STRING(5701,"Celcius"),
    // some entries not shown……. .
   LWM2M_RESOURCE_FLOATFIX_VAR(5602,
                      &max_temp));
```

An instance will be created and included in an object by:

```
LWM2M_INSTANCES(temperature_instances,
   LWM2M_INSTANCE(0,temperature_resources));
   LWM2M_OBJECT(temperature,3303,
                temperature_instances);
```

This is followed by a call to add this object to the engine's static array of lwm2m_object_t pointers:

```
  lwm2m_engine_register_object(&temperature);
```

C. HPP Mapping of OMA LWM2M on Contiki

1) Mapping the relevant Object Structures

A set of header files with static definitions for IPSO resources (properties in HPP) and objects were created, e.g.

```
  #define IPSO_Sensor_Value_PROP_ID 5700
  #define IPSO_Generic_Sensor_OBJECT_ID 3300
```

A static definition of the IPSO Classes was created in a header file, initialised as an array of li_class_property_t to hold the property names and types of the class. For example, a Temperature Sensor class is defined as:

```
  //name,property_id,type,mode,permission
  li_class_property_t IPSO_Sensor_Value =
    {IPSO_Sensor_Value_PROP_NAME,
     IPSO_Sensor_Value_PROP_ID,
     real32, DYNAMIC, READONLY};
```

These properties are grouped in li_objects with valuelists of key value pairs of property id, length, value, next (or callbacks to set values) as in:

```
li_kv_entry_t tSensor_vals[] = {
  // property id, length, value, next
  {IPSO_Sensor_Value_PROP_ID,4,"0",
                        &tSensor_vals[1]},
  // some entries not shown…….    .
  {IPSO_Sensor_Type_PROP_ID,12,"Temperature",
                        &tSensor_vals[9]},
  {LI_END_PROP_ID, 0, NULL, NULL}
};
```

The classes implemented in a given node are added to an array of li_class_t to define the properties (by pointing to that list) and the relevant callbacks as below and the getter/setter callbacks per property must be coded to access dynamic values such as sensor readings:

```
li_class_t node_classes[] = {
  {IPSO_Generic_Sensor_OBJECT_ID,
  HPP_PREFIX,
  IPSO_GenericSensor_PropCount,
  &IPSO_GenericSensor[0],
  0, NULL,&localFunctions},.......
};
```

Then the getter/setter callbacks per property must be coded to access dynamic values such as sensor readings.

2) Integrating the HPP Objects into the LWM2M engine

The existing LWM2M context and REST code were retained and the objects in the object space were available over the existing REST interfaces. This made testing easier by using the Copper Browser plugin and Leshan server for OMA as for any OMA LWM2M node.

lwm2m_engine_register_default_objects() was extended to call *dm_service_initialise(),* which in turn sets up a DM_SOURCE_SRV with *service_source_init()*. This initialises the li_node information and calls *dm_li_add_class()* and *dm_add_li_instance()* to add the supported DM service, node and local instrumentation classes and instances to the object space. This used a call like

```
  rv = addInst(this_info_ptr,
               &myGenericSensorInstances[0],
               IPSO_Generic_Sensor_OBJECT_ID,
               "0", IPSO_Keys_PropCount,
               IPSO_Generic_Sensor_INDEX);
```

lwm2m_engine_handler() was changed to use Data Model calls such as dm_find_instance_by_name() to get, set LWM2M resources, returning REST responses as before.

V. Review of Implementation

This section considers the suitability of our architecture based on our implementation of the OMA LWM2M model.

1) Implementation Complexity

The code extracts show the mapping of IPSO resources to our object classes and instances to be straightforward and that it was simple to integrate the OMA Engine with our data model service layer and object space.

2) Memory Use

The table below indicates the memory use in bytes of our components compared to the existing ipso, rest and coap code on a Wismote WSN node running Contiki including IPSO LightControl, Generic Sensor DigitalInput objects. This shows the size of our layers is suitable to run on constrained nodes.

Component	Code Size	Data Size
Lwm2m and OMA	7742	3807
Rest+Coap	8278	2228
Dm_ and li_ layers	1686	1316
Object layer (and supporting utils)	1036	139
IPSO extensions for dm_ and li	4126	1684
Full Stack	60474	21753

Fig. 5. Memory Use on Contiki

3) Abstractions

The holistic architecture does not require specific middleware nodes or servers such as the OMA Leshan server used here to retrieve/set OMA data. The existing LWM2M implementation on Contiki3.0 maps object/instance/resource nicely using structures. The use of a static array to hold pointers to the instances is hidden by methods like *lwm2m_engine_register_object()*.Similarly, the implementation of the object space, is hidden by the object library API. The value of the data model service role abstractions has been shown to a limited extent as only the DM_SOURCE_SRV role was implemented. The integration here shows a fuller integration of the HPP protocol with the IPSO and OMA code would take advantage of the ability of our architecture to store and cache data from remote nodes by adding the DM_STORE_SRV role for LWM2M data from remote nodes. LWM2M does have the concept of registration to one or more servers, which includes objects, but this does not appear to be as rich as the defined DM_ roles. The use of the data model service layer allows a much richer matching in a request than OMA LWM2M as it can match on template or wildcards or particular properties.

Also, LWM2M uses the URI of objectid/instance or objectid/instance/resourceid to select a resource, whereas our data model service distinguishes key and non-key properties in

the class, which allows the straightforward implementation of other data models.

4) Object and Property Mapping

The implementation has shown that a per property based approach fits naturally with how the low level functionality is often performed on devices, e.g. with a GPIO call per property. It also allows selection of only the implementable attributes on a node, so saving memory per implemented class. Both LWM2M and HPP support this approach. While REST resources such as led and sensors generally have a few properties, the IPSO Application Framework[19] defines function sets as groupings of individual attributes, e.g. a device at /dev has 12 resources, e.g. Manufacturer at /dev/mfg.

5) CIM vs OMA LWM2M

Comparing to our earlier implementation of the Common Information Model (CIM)[5], the per property(resource) data model of LWM2M is more suited to constrained devices, e.g. the IPSO Generic Sensor definition is much simpler than the inheritance involved in constructing a CIM_NumericSensor. CIM also uses lots of strings, e.g. for names, which is expensive in memory, even if only stored once as in HPP. The IDs in IPSO are easier to program and more efficient in memory. IPSO also has fewer types than CIM, as suits constrained devices.

CIM has specific object methods, whereas IPSO uses resources with implied actions, e.g. CIM has setAlarmState which can be used to set a led, whereas IPSO Light Control uses the simple On/Off (5850) boolean Actuator resource. In this case, the IPSO approach is simpler. It is less obvious in resources like “Reset Min and Max Measured Values”, while the implied use of “On-Time”(5852) or “Off-Time”(5853) to reset is not consistent with specific Reset resources elsewhere.

6) Mapping LWM2M Resources

The defines used in the HPP header files were easy to generate from the IPSO docs by substituting “_” for “ “, but some issues were caused by certain characters or mixed capitalization, e.g. use of “/“ in On/Off (5850), Off-Time(5853) and Minimum Off-time(5525). Using digits for OIDs reduces string usage and is suitable for M2M, but it is more user friendly to use a RESTful well-known URI, so we also allowed names, e.g. /Device/0/Manufacturer as well as 3/0/0.

VI. Conclusion

This paper has shown our architecture’s novel design point of a tuple-based object store for both local and remote node data with a simple API allowed a data model service layer and local instrumentation layer to hold IPSO objects without needing special middleware nodes. From this it follows that it would be straightforward to add objects remotely, illustrating the potential of the architecture and service abstractions. The object definitions used for our local instrumentation layer and its per property approach mapped well to the resource approach in IPSO and the underlying Contiki hardware libraries.

Following our earlier implementation of a CIM data model, this implementation of the OMA model on constrained nodes running Contiki3.0 OS shows the benefit of using a data-centric approach, such as using our architecture, for both local and remote node data. For this reason, we strongly recommend that IoT architectures should revolve around data, with abstractions to represent the node’s local data and the capabilities of nodes and applications, especially as models such as OMA LWM2M are developed further.

This implementation of the LWM2M data model in our architecture allows further work to consider the value of storing remote node data in the object space and how our HPP protocol should interact with CoAP’s observe and caching facilities.

References

[1] S. Haller, “The Things in the Internet of Things”, Internet of Things Conference, 2010

[2] M. Koster, “Information Models for an Interoperable Web of Things”, Position paper for W3C Workshop on the Web of Things–Enablers and Services for an Open Web of Devices, June 2014.

[3] G. Fortino et al, "Towards a Development Methodology for Smart Object-Oriented IoT Systems: A Metamodel Approach.", IEEE International Conference on Systems, Man, and Cybernetics (SMC), 2015

[4] Open Mobile Alliance, “Lightweight Machine-to-Machine Technical Specification v1.0”, December 2015

[5] D. Tracey, C. J. Sreenan, “A Holistic Architecture for the Internet of Things, Sensing Services and Big Data”, Data-intensive Process Management in Large-Scale Sensor Systems, CCGrid 2013

[6] R. Fielding, “Architectural Styles and the Design of Networkbased Software Architectures”, Doctoral dissertation, University of California, Irvine, 2000

[7] J. Jimenez, M. Koster, H. Tschofenig, “IPSO Smart Objects”, IPSO Position paper for IOT Semantic Interoperability Workshop, 2016

[8] M. Kovatsch, S. Duquennoy, A. Dunkels, “A Low Power CoAP for Contiki”, IEEE 8th International Conference on Mobile Adhoc and Sensor Systems (MASS), 2011

[9] Wei-Gang Chang, Joseph-Lin Fuchun, “Challenges of incorporating OMA LWM2M gateway in M2M standard architecture”, IEEE Standards for Communications and Networking (CSCN), 2016

[10] S. Rao, D. Chendanda, C. Deshpande, V. Lakkundi, Implementing LWM2M in Constrained IoT Devices”, IEEE Conference on Wireless Sensors (ICWiSE), 2015

[11] P. Costa, L. Mottola, A. L. Murphy, and G. P. Picco. “Programming wireless sensor networks with the TeenyLIME middleware”, Proc. of the 8th Int. Middleware Conf., 2007

[12] D. Gelernter, “Generative communication in Linda”, ACM Transactions on Programming Languages and Systems (TOPLAS), Volume 7 Issue 1, Jan. 1985

[13] M. Ceriotti et al, “Monitoring heritage buildings with wireless sensor networks: The Torre Aquila deployment”, 8th Int. Conf. On Information Processing in Sensor Networks (IPSN), 2009

[14] Gian Pietro Picco et al, "LighTS: A Lightweight, Customizable Tuple Space Supporting Context-Aware Applications", ACM Symposium on Applied Computing, 2005

[15] C. Scholliers, E. Boix, W. De Meuter,, "TOTAM: Scoped Tuples for the Ambient", Proceedings of the Second

International DisCoTec Workshop on Context-aware Adaptation Mechanisms for Pervasive and UbiquitousServices, 2009

[16] S. R. Madden, 'The Design and Evaluation of a Query Processing Architecture for Sensor Networks", Ph.D. Thesis. UC Berkeley, 2003

[17] C. Intanagonwiwat, R. Govinden, D. Estrin, J. Heidemann, F. Silva, "Directed Diffusion for Wireless Sensor Networking", IEEE/ACM Transactions on Networking, Vol 11, No. 1, February 2003

[18] D. Tracey, C. J. Sreenan, "CacheL - A Cache Algorithm using Leases for Node Data in the Internet of Things", IEEE Future Internet of Things and Cloud (FiCloud) 2016

[19] Z. Shelby et al, "The IPSO Application Framework", Internet-Draft, draft-ipso-app-framework-04, August 2012