Toward A Common Modeling Standard for Software Update and IoT Objects

BY NED SMITH (ned.smith@intel.com**)**

Abstract

IoT object models (e.g. IPSO[i], LWM2M[ii], OCF[iii], Alljoyn[iv]) often apply information modeling techniques that enrich model definition semantics, enable syntax checking, code generation and automated mapping between data modeling languages and data serialization formats. These conventions for object modeling facilitate interoperability and automation and are desirable properties for software update infrastructure. Efficiencies in system design, complexity and cost can be improved by relating them under a common modeling approach for constrained environment devices. Security and interoperability benefits are also realized.

1. Terminology

Information model[v] in software engineering is a representation of concepts and the relationships, constraints, rules, and operations to specify data semantics for a chosen domain of discourse. Typically it specifies relations between kinds of things, but may also include relations with individual things. It can provide sharable, stable, and organized structure of information requirements or knowledge for the domain context.

Data modeling[vi] in software engineering is the process of creating a data model for an information system by applying formal data modeling techniques.

Data modeling language[vii] is a modeling language for describing general text and binary data in a standard way. A DFDL model or schema allows any text or binary data to be read (or "parsed") from its native format and to be presented as an instance of an *information set*. Information set is a logical representation of the data contents, independent of the physical format.

Serialization[viii] is the process of translating data structures or object state into a format that can be stored (for example, in a file or memory buffer, or transmitted across a network connection link) and reconstructed later in the same or another computer environment. When the resulting series of bits is reread according to the serialization format, it can be used to create a semantically identical clone of the original object. For many complex objects, such as those that make extensive use of references, this process is not straightforward.

2. IoT Object Modeling

Object modeling serves an important role in IoT systems by capturing semantic and syntactic constraints in an abstract representation so that IoT systems are flexibility, interoperable, reliable and secure. Object modeling can be decomposed into 4 layers having an information model, data model, data model serialization and security. For example, a bridge between dissimilar IoT networks may rely on information and data models to ensure object syntax and semantics are correctly translated.

The left side of figure-1 shows a 4-layer IoT object model.

Information Model (IM)

An Information Model captures object semantics. There are many IM languages (IML), such as UML, OWL or RDF. IM languages facilitate automation that helps both human and machine richly describe and process model semantics. IM expressions captured using an IML are more easily mapped from one IML representation to another thereby facilitating interoperability across IoT networks. It also facilitates translation into a data model.

Data Model (DM)

The Data Model (DM) defines data syntax. DM schemas marry IM semantics with DM syntax enabling automated object syntax checking and code generation. Objects can be verified according to the DM schema at critical points in object lifecycle. For example, syntax checks at object instantiation and prior to or following serialization ensures object processors do not parse maligned structures that may produce undefined behavior. There are many DM languages to chose from, for example, JSON schema, XML schema, RAML[ix] – (this list is not exhaustive).

Objects expressed using a DM improve interoperability between dissimilar IoT networks when connected via an object translation bridge. Object translation tools are more easily implemented.

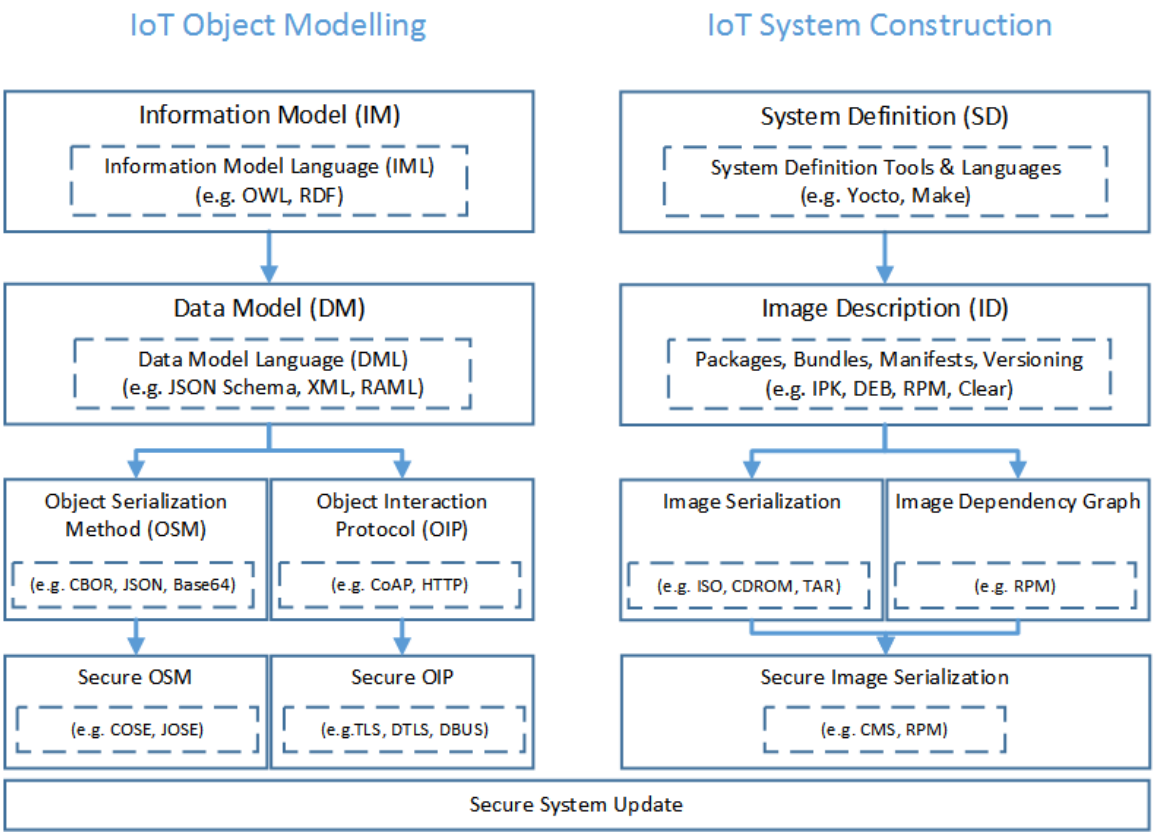

Figure 1 – Comparison of IoT object modeling techniques with IoT system construction techniques. Secure system update is a common element.

Object Serialization Method (OSM) and Object Interaction Protocol (OIP)

The Object Serialization Method (OSM) specifies how objects appear when converted to wire formatting. Objects contain both data values and metadata. There are many serialization formats; ASN.1[x], JSON[xi], CBOR[xii], Base64[xiii], XML[xiv] just to name a few.

Object Interaction Protocol (OIP) specifies object interfaces and interaction semantics that ensure objects can respond to and effect environmental changes in a controlled way.

Secure OSM (SOSM) and Secure OIP (SOIP)

Security objectives often include authentication, authorization, access, accounting, confidentiality, privacy, consistency and availability and are concerned with making appropriate trade-offs. To these ends, the information and data models should inform security.

Security introduces its own data, semantics and syntax used to describe security objects that implement security, privacy, consistency and availability strategies.

Security objects interface with data objects at the SOSM / SIOP layer in order that the various protection strategies are efficacious. This is most effective when security system design and information system design occur in concert with one another.

For example, personal health systems containing sensitive medical information may require encryption to protect privacy, but may also require override to inform first responders in the event of a medical emergency.

In practice, the OSM and SOSM layers are recursively interdependent. For example, an agricultural soil moisture sensor may inform water distribution at pumping stations, water quality controllers and commodities brokers where each entity has a distinct and self-interested security objectives. Cost and performance constrained endpoint sensors depend on intermediate processors that enforce the manifold security objectives by layering protections over data objects that may have previously layered protections.

For these reasons, it is highly desirable for security modeling and information-modeling disciplines to have IM, IML, DM, DML, OSM, OIP tools and techniques in common.

3. IoT System Construction

The right side of Figure 1 offers a decomposition of a device construction taxonomy that produces 'images' of executable or interpretable objects. Files are organized into packages, bundles and archives that represent installable images used during software update. Secure distribution and installation ensures software update images are not an attack vector for IoT networks. Secure boot procedures may re-assert the security properties established at update time still hold. Hence, it is reasonable that software update and secure boot mechanisms be treated together.

System Definition (SD)

System Definition (SD) is arguably the first step in a software update mechanism because it uses an information model to capture an abstract representation of the system, platform or device. The Yocto Project[xv] is a methodology for defining and building IoT systems. It employs a rudimentary system definition language that describes buildable software objects and their interdependencies. For example, *Bit Bake* uses a *recipe* file (.bb) to enumerate various kernel drivers and modules. A catalogue of Bit Bake recipes can be used to automate construction an *off-the-shelf* IoT device.

Image Description (ID)

System Definition produces an image description (ID) file that describes bundles[xvi], packages[xvii] and manifests[xviii] that relate compiled and interpreted software images in accordance with a versioning scheme. Image description informs quality control procedures (i.e. testing, validation, release) so that configuration management can track regressions and security events.

Image Serialization (IS) and Image Dependency Graph (IDG)

Image Serialization (IS) describes how installable images are prepared for transport and installation on a target device. This may include, for example, construction of an ISO, CDROM or TAR archive. Image serialization allows flexibility in optimizing the most effective approach to distribute the software update image to the target platforms.

Image Dependency Graph (IDG) is a structure that tracks code execution paths identifying files, packages and bundles that are needed to ensure successful execution. IDG also aids site-specific customization. Unneeded features may be omitted or specialized features may be included. For example, a general-purpose IoT-bridge may handle HVAC, lighting and surveillance traffic. But a site-specific deployment only enables HVAC functionality. During a software update scenario, a site-specific policy informs the install tool that selects the HVAC subset of objects from the software update image.

Secure Image Serialization (SIS)

Secure Image Serialization (SIS) protects the serialized image according to a security policy. Security goals may include image authentication, encryption and integrity protection. A security model informs the image serialization regarding trust, credentials and protocols for protecting serialized images. For example, CMS[xix] is a data model, based on (ASN.1) and DM serialization method (BER[xx]) for describing security information. RFC4108[xxi] is a proposed standard that further uses ASN.1 and BER to describe a manifest structure useful for protecting firmware packages.

RPM[xxii] defines a *manifest* structure that includes image integrity checking. However, it lacks confidentiality, authentication and access control support. Integrity strength of function may be lacking comprehensive support for NIST Suite-B[xxiii] recommendations. RPM and its cohorts IPK, DEB and Clear may be a poor choice as a comprehensive DM for object definition because they lack much of the richness that a classical DM (i.e. XML, JSON, RAML) provides. But one might argue they specialize in an area that requires specialization.

Software update anticipates manifest structures that relate protection mechanisms with complex update image structures consisting of files, packages, bundles and other metadata. Software update tools must navigate complex IoT network topologies and cross multiple security boundaries. Site-specific customization implies install tools can interpret localized security polices and deployment considerations. Consequently, the manifest structure is only one element of a comprehensive software update solution.

Observations

IoT object definition that applies a 4-layer information model seems well suited for effectively dealing with complex interoperability and security challenges. Similarly IoT image construction and distribution strategy that applies a 4-layer information model seems well suited for dealing with complex device deployment challenges.
Security enforcement is a point where data objects and execution objects meet. Protection strategies for both data and execution objects are similar. Both expect a rich environment for describing IoT devices, networks and objects. Both anticipate consistently applied security semantics and syntax.

4. Recommendation

Secure operation in a constrained environment is an overriding theme in IoT. Reducing the number of parsers, syntax checkers and semantic validators is a consideration when trying to minimize a device footprint. To that end, the architecture of Figure 2 is suggested.

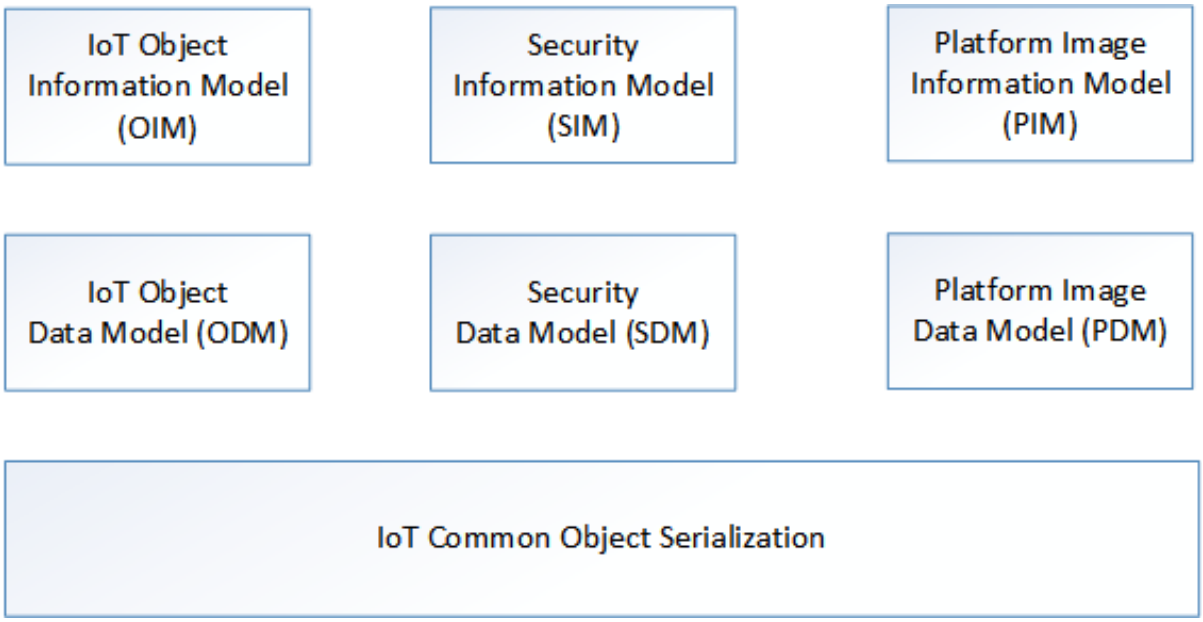

Figure 2. – A common IoT object serialization method that allows IoT data objects, platform image objects and security objects to be expressed using the same serialization suggests a constrained device implementation need not require multiple serialization conventions.

For example, given an IPSO Alliance object information model[xxiv]; RPM as the platform image information model and CMS as a security information model. Even if XML is the data model language used to describe IPSO objects; ASN.1 is used to describe software update packages and JSON is used to describe security data. This architecture would require serialization under a common object serialization standard – for example CBOR. Consequently, the device firmware needed to process a secure software update may be leveraged when processing IoT objects during normal operation.

Security and reliability benefit is also achieved because testing and validation of firmware also benefits IoT object processing. Overall complexity is reduced resulting in fewer bugs and therefore fewer updates to deployed devices.

Reducing the need for software updates while still providing software update mechanisms in IoT systems is a strategy that scales best given projected growth[xxv] of IoT networks in the next 4 years.

Software update mechanisms that are defined with a common object serialization layer will benefit from a richer set of authoring and modeling tools that can be used to describe more completely system behavior for both data and executable as well as describe how both are secured. Secure system architecture requires consistent

application of security practices. This is more easily achieved when both data and executable objects use common understanding of security encodings.

---

[i] IPSO - http://www.ipso-alliance.org/wp-content/uploads/2016/01/ipso-paper.pdf
[ii] LWM2M - http://technical.openmobilealliance.org/Technical/technical-information/release-program/current-releases/oma-lwm2m-object-connectivity-management-v1-0
[ii] LWM2M - http://technical.openmobilealliance.org/Technical/technical-information/release-program/current-releases/oma-lwm2m-object-connectivity-management-v1-0
[iii] OCF - http://openconnectivity.org/wp-content/uploads/2016/01/OIC_Specification_Overview_201501131.pdf
[iv] Alljoyn - https://allseenalliance.org/framework/documentation
[v] https://en.wikipedia.org/wiki/Information_model#cite_note-Lee99-1
[vi] https://en.wikipedia.org/wiki/Data_modeling
[vii] https://en.wikipedia.org/wiki/Data_Format_Description_Language
[viii] https://en.wikipedia.org/wiki/Serialization
[ix] RAML - docs.raml.org/
[x] RFC6025 ASN.1 - https://tools.ietf.org/html/rfc6025; ASN.1 - ITU X.680 - http://www.itu.int/itu-t/recommendations/rec.aspx?rec=x.680
[xi] RFC7159 JSON - https://tools.ietf.org/html/rfc7159
[xii] RFC7049 CBOR - https://tools.ietf.org/html/rfc7049
[xiii] RFC4648 BASE64 - https://tools.ietf.org/html/rfc4648
[xiv] XML - http://www.w3.org/TR/REC-xml/
[xv] Yocto Project - https://www.yoctoproject.org
[xvi] Bundles - https://clearlinux.org/documentation/index_bundles.html
[xvii] Packages - https://en.wikipedia.org/wiki/Package_manager
[xviii] Manifest - https://en.wikipedia.org/wiki/Manifest_file
[xix] CMS – RFC5652 - https://tools.ietf.org/html/rfc5652
[xx] BER – ITU X.690 - http://www.itu.int/ITU-T/studygroups/com17/languages/X.690-0207.pdf
[xxi] RFC4108 - https://tools.ietf.org/html/rfc4108
[xxii] RPM - https://en.wikipedia.org/wiki/RPM_Package_Manager
[xxiii] NIST Suite-B - http://csrc.nist.gov/groups/SMA/ispab/documents/minutes/2006-03/E_Barker-March2006-ISPAB.pdf
[xxiv] IPSO Alliance object model - https://www.iab.org/wp-content/IAB-uploads/2016/03/ipso-paper.pdf
[xxv] 200B IoT devices by 2020 - http://www.intel.com/content/www/us/en/internet-of-things/infographics/guide-to-iot.html