A Modular Open Source Platform for Web Scale IoT Interoperability

Michael J Koster
October 19th, 2013

Interoperability for the Internet of Things

Previous articles in this series discussed architectures and approaches to provide meaningful interoperability. Protocol interoperability allows any application to interact with any connected thing using any M2M protocol.

Interoperability also provides for reuse of software components, allowing a common platform and tool set across diverse use cases. It should be easy to integrate data and things from diverse sources into a single application. Diverse UI platforms and new UI/UX models should be enabled to be easily integrated into systems.

Figure 1 shows how a platform for interoperability would allow applications to discover and connect to diverse things using a choice of M2M protocols.

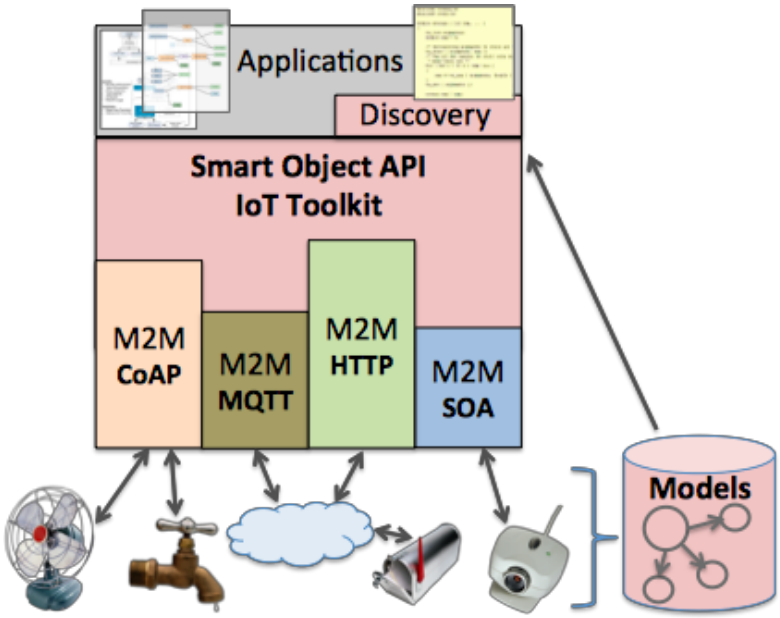

Figure 1 - Interoperability

Data models enable machine understanding independent of M2M protocols

Data models describe connected things to applications, driving discovery and linking, and allow abstraction of M2M protocols. The effect is similar to the separation of the data plane from the control plane in software defined networking.

In figure 2, a Semantic Proxy provides a common representation of data models that originate in different catalogs and repositories. Because each application or resource endpoint is using a common data model, any M2M protocol can be used to update the sensor or thing data. The common data model describes enough about the endpoints to allow abstraction of the transport mechanism.

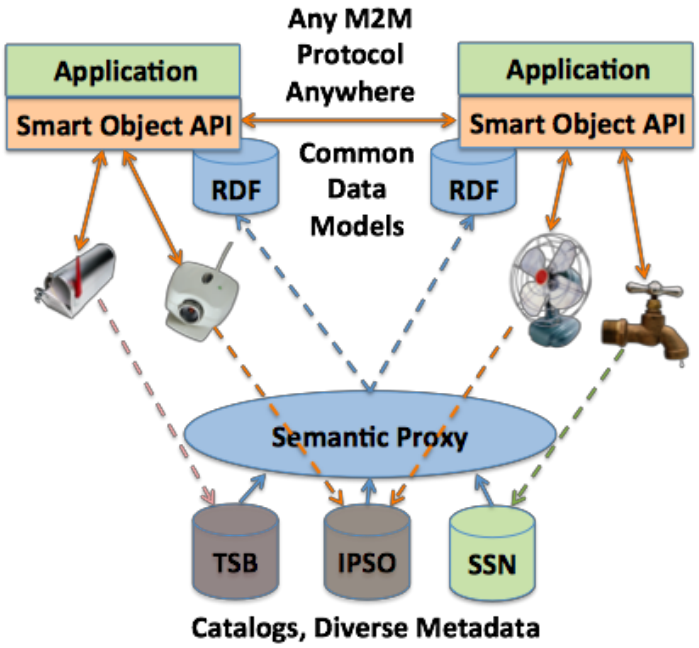

Figure 2 - Data Models enable M2M abstractions enable M2M Interoperability

It should be noted that the semantic proxy consists of mainly agreed-upon bindings of relations (predicates) and attributes (objects) and representation formats that allow each application to conveniently use it's favorite catalog and representation format while interoperating with others.

Open Source Software for IoT is becoming common

Recent developments in open source IoT components are starting to enable an ecosystem as well as a modular open source IoT software stack.

- The Mosquitto MQTT broker and client are robust, full featured components.
- Node-RED is a new visual programming tool for event-driven software composition.
- CoAP stack software is available in several forms to integrate into different endpoints.
- RDFLib provides a robust linked data endpoint.

- Neo4J is a graph database that provides a full SPARQL endpoint with scalable discovery potential.
- The Dojo UI toolkit is aligned with HTML5 and can provide robust multi-context multi-screen UI models.

IoT Toolkit can provide a common object model and data model framework to enable endpoint discovery and transport abstraction as outlined earlier.

There are now enough components to enable IoT Toolkit to be a component of a larger system based on an **Open Source IoT Stack,** and focused on Interoperability and Web Standards.

Model-View-Controller Structure in IoT Applications

The Internet of Things includes systems of feedback control, where *observations* of key attributes provide *information* that eventually creates a change in the system through *actuation*.

This cycle of Observation => Information => Actuation creates a *feedback control loop* where the observed property is controlled in a system known as a *closed loop*.

Figure 3 shows the typical structure of some closed feedback loops in IoT systems. There are *autonomic feedback loops* which involve only software making decisions. An example of this is a motion sensor turning on a light.

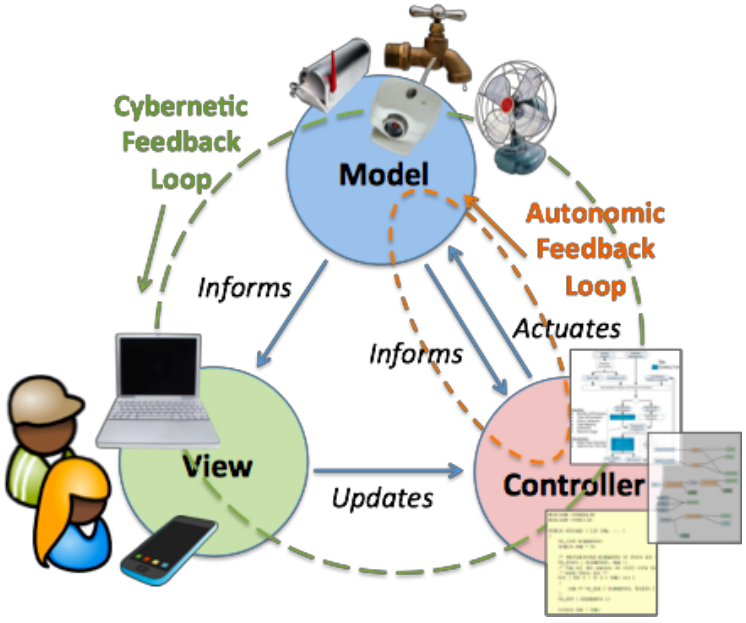

Figure 3 - IoT Control Loops and the Model-View-Controller macro-pattern

There are also *cybernetic feedback loops*, which involve a person in the loop participating in making decisions. In this example, the person in the cybernetic loop is updating settings of an autonomic loop controller.

In the above example, the **Model** is a representation or an abstraction of the physical things and their attributes, which *informs* a **Controller**. The Controller is a piece of software which makes *actuation* decisions based on the information, and sends actuation commands to the thing using it's modeled affordances. The software goal is to maintain a desired state of the thing through it's model.

In the above example, people interact with the system through a **View**, which is constructed based on *information* from the model. By observing information presented in views, people make conclusions about the state of the things and consequently make control decisions from which they *update* the controller.

Some **basic mapping of components to a M-V-C macro pattern** are shown in figure 4.

IoT Toolkit provides an object encapsulation of the data **models** and event models in the Smart Object API, and enables connection to physical objects. IoT Toolkit provides scalable resource discovery, adapts to diverse data formats and protocols, provides storage and persistence of objects and data streams, and facilitates a model-driven binding of physical objects to URLs.

Node-RED allows resources from IoT Toolkit Smart Objects to be wired directly to software handlers and other resources, for example actuators, displays, and analyzers. The internal event model of Node-RED roughly corresponds to MQTT, and as such makes a natural connection to the MQTT Observer resource of the Smart Object API.

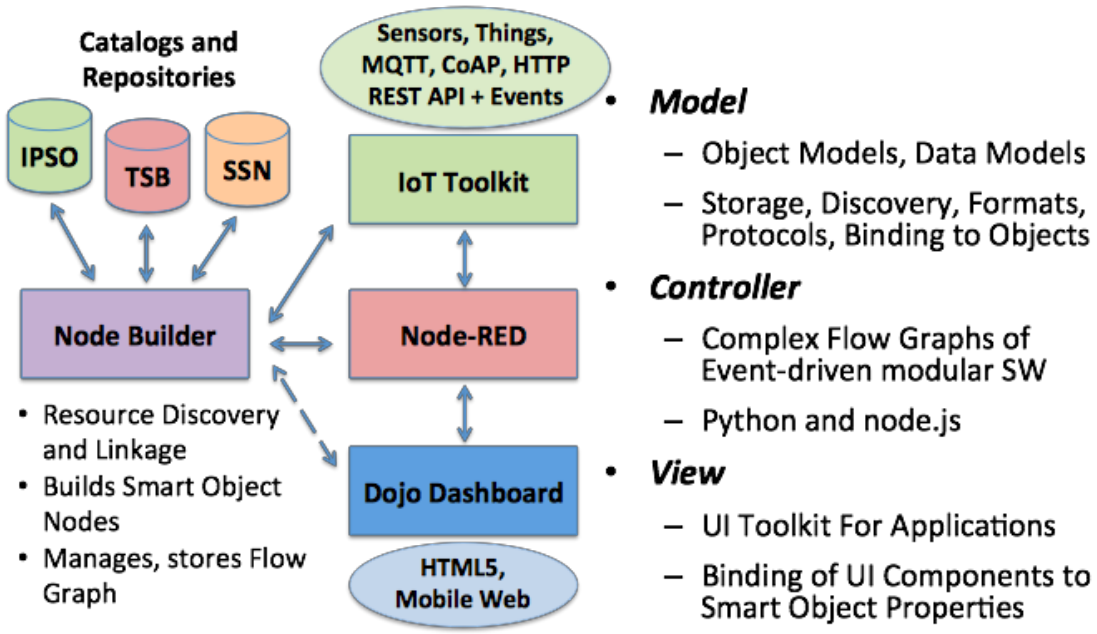

Figure 4 - Mapping of Software Components to the M-V-C Macro Pattern

The *Node Builder* is a new component that will discover Smart Object and other resources and data sources, and create Node instances from them that can be wired into Node-RED flow graphs. The Node Builder can use the Semantic Proxy to search and link information from multiple diverse *catalogs and repositories* into event driven applications. The Node Builder will also provide APIs to facilitate the automatic and run-time use of discovery and linkage to build ad-hoc graphs based on real-time context.

Dojo Toolkit is an open source UI and application system that builds Web scale and mobile application components based on a UI toolkit. The Dashboard builder is another new component that enables Widgets and controller components based on Dojo Toolkit to be wired into Node-RED flows and organized onto user screens. Information from the data models of things will provide units, scale and qualititative interpretation metadata for controlling the presentation on diverse devices and for different use cases.

Model-View-Controller Workflow in the creation of application graphs is depicted in Figure 5.

The **Node Builder** discovers resources and data sources from catalogs and existing data models according to the requirements of the application. For example, if the application is a Smart Thermostat, the data sources would include temperature in the room being controlled, a control for the heating device in the room, information about windows and solar gain, local weather and weather forecast, and current and predicted room occupancy.

The developer then discovers, selects, or creates application software components and UI components to build the application behavior and UI from.

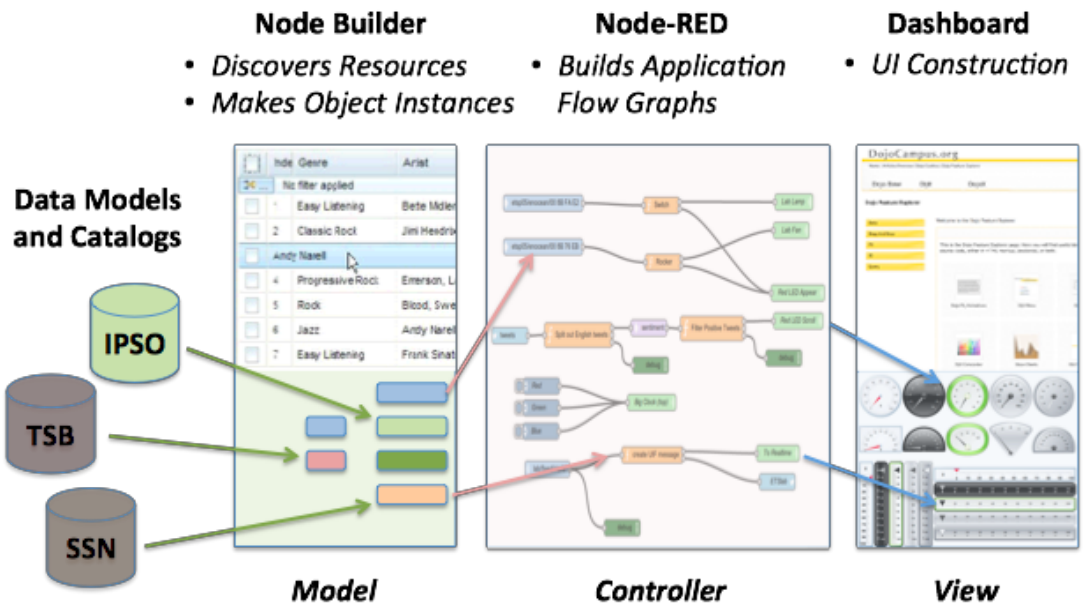

Figure 5 - Application Development Tools and Workflow

The Application developer then creates a graph. The resources and application components can be either wired together manually by drawing arcs with the tool, or components can be automatically assembled by the Node Builder based on a database of models (ML system) or user templates, etc. Outputs are connected to actuators or UI components which can be selected from the **Dashboard** builder.

At the end of the process, an application graph is deployed by **Node-RED** which connects the resources, creates a UI instance, and begins operating on the datastreams and events.

The Run-Time Architecture of the deployed system is shown in figure 6.

Diverse protocols are used to apply the most suitable M2M system for a given connection, whether from cloud to cloud, sensor to gateway, sensor to cloud, gateway to cloud, or data models being accessed from a catalog.

Data models from various catalogs are referenced to create a running instance of an application, and are used occasionally by the running application to discover or adapt to new contextual information.

A Local Control Gateway provides an always-on node for low power sensors to connect, and runs local applications for observing and controlling water flow and air movement. Other sensors connect to both gateway and Personal cloud service, and some sensors connect directly to cloud services.

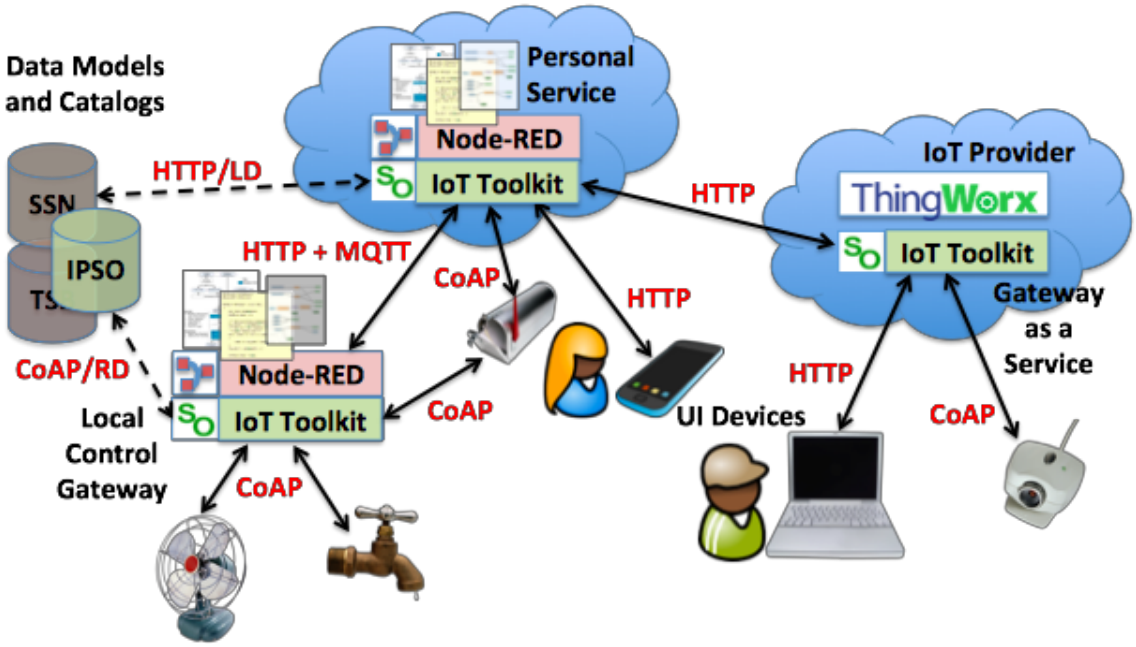

Figure 6 - Example of Run-Time Deployment

An instance of IoT Toolkit is shown deployed as “Gateway-as-a-Service” to connect a service provider, in this case ThingWorx, into the system to observe and expose it’s resources.

User Interface devices can connect to cloud services or directly to gateways, though there is likely more hardware service capability available in services than gateways at present.

IoT Toolkit instances in the cloud service and gateway can both run application software, providing a local backup capability if internet service is disrupted, or for better response time, or for availability of hardware e.g. GPUs for analytics. Many gateways and services can co-operate to provide layers of control and analysis.

Building an Open Stack for the Internet of Things

The components for creating a robust end-to-end Open Source IoT stack, from sensor to web application, are now becoming available. As more of the use cases become understood and addressed, this will build out in several key areas.

By adding a few new components to glue application graphs together from resources, we can create web scale interoperable applications that can be distributed across local application gateways and cloud services.

Existing components can be integrated at web scale, using URIs to bind connections, to create a macro instance of the well-known Model-View-Controller pattern, which allows for parallel autonomic feedback loops and cybernetic feedback loops.

The Model-VIew-Controller macro pattern provides a framework for the structured division of responsibility between people and software in IoT applications. It also provides a framework for high level interoperability between data sources, control elements, and UI elements.

Such a stack can form the basis of a new service infrastructure, which can provide Platform-as-a-Service functionality while avoiding the single points of failure and lock-in typically associated with PaaS. Open Source PaaS can provide the mechanism for fault-tolerant run-anywhere application platform functionality needed by IoT applications as we begin to depend on that functionality more and more in our daily lives.

http://www.slideshare.net/michaeljohnkoster/open-source-stack-for-io-t