Use of Introspection for Facilitating Interoperability

Dave Thaler <dthaler@microsoft.com>

Summary: In this paper, we use the term "Introspection" to mean the ability to dynamically retrieve a machine-readable description of a given device's schema (which term in this paper includes both the syntax and the interaction model). Whether such introspection is supported today, and the extent of such support, varies greatly across IoT ecosystems today. This paper summarizes the philosophy behind the semantic interoperability work within the Open Connectivity Foundation's technologies (including the UPnP, AllJoyn, and OCF protocols) where introspection has played a key role, and argues that extending such capability in other ecosystems would provide similar benefits and allow greater interoperability between IoT ecosystems in general.

Introspection Use Cases

Machine-readable schemas for resource or device types can be (and are) used for the following purposes, given a schema for a resource or device type:

1) Code generation: generate a strongly-typed client-side library for talking to such resources. Strong typing greatly eases the burden of application development and debugging, and can also improve efficiency since operations such as validation can be done on the client.
2) Code generation: generate server-side stubs for implementing such resources.
3) Code generation: generate tests for compliance.
4) Code generation: generate fuzz tests.
5) Universal-remote-like generic UI: "object browsers" are common in various other contexts such as management tools. Schemas can be used to construct, at runtime, generic user interfaces for monitoring or controlling IoT devices. Such browsers are most typically used by advanced users such as developers or troubleshooting personnel, but if introspected information also includes UI markup information (e.g., arrangement of fields, type of UI control, etc.) then it can also be used by end users, just as browsers are used to render web pages.
6) Algorithmic mapping: allow a gateway (or "bridge") to algorithmically map resources from one ecosystem into "vendor-specific" (i.e., custom) resources of another. We will return to this later below.

For example, one of Microsoft's scenarios that it provided developer tools for AllJoyn was as follows. A developer wants to create his own client application that can interact with an arbitrary device he just purchased and put on his network. He brings up a development environment (Visual Studio in this case), and in less than 5 minutes, he has a compiled strongly-typed library he can use for interacting with that device, including any vendor-specific functionality of that device.

Querying current state is not enough

Most IoT protocols support querying the current state of a device/resource in a self-describing way. However, this is not the same thing as introspection. That is, the *current* state is not sufficient to express *capability.* For example, say a device supports the ability to be configured with a collection of data (e.g., scenes, autonomous rules, ACLs, etc.) The current state of such a collection might be empty. The schema, on the other hand, would express the fact that entries can be created, and what properties entries can have, and what values properties can have.   Even global scalar properties need to have expressed what values are legal, the units, a human-readable description, etc.

Schema repositories are not enough

Often such schemas are merely specified in specifications, without any repository other than a web site holding the specification documents. Sometimes they may be present in some repository such as oneiota.org or schema.org, but (especially for ecosystems that allow vendor extensions) there may be no programmatic way to get the appropriate location to find the machine-readable schema for a given resource.   Finally, even where such locations are resolvable, they often require Internet connectivity and so may be useful for code generation at development time, but may not be usable for run-time scenarios (e.g., algorithmic mapping by a bridge, generic UI, etc.) in an environment that is not connected to the Internet.   The most widely useful cases are where each device itself can supply its own schemas on demand to a client.   The limitation however is that the device must have the storage available to hold such schema, and hence this can be difficult for more constrained devices.   As such, we argue that **devices should minimally be able to supply to a client at runtime the *location* of the schemas it implements**, whether that location be hosted on the device itself (and thus usable by protocol gateways inside disconnected networks), or on other servers across the Internet.

Standardizing schemas is not enough

There are a couple reasons why the schema needed for a given device might differ from the schema developed by a standards organization.   First, the standard schema might include both mandatory and optional components (e.g., optional properties, or optional-to-support values of certain properties), so the schema relevant to a specific device will specify what that device supports and not what it does not support. Second, the device might also support vendor-specific extensions that have no standard schema.   IoT is still early enough in the innovation cycle that limiting devices to only support standard schemas is too limiting.

Machine-readable schema syntax is not enough for all purposes

While machine-readable schemas are important and may suffice for a few use cases, they are not sufficient for some of the uses cases listed above. Specifically, use cases also need human-readable descriptions.   The first four use cases listed above are development-time use cases and in such use cases, descriptions are consumed by developers, e.g., by code generators using description text to generate human-readable comments in generated code, where (say) English might be a common expectation.   Runtime use cases, on the other hand, are more generally usable by non-English users. Thus, to enable such cases, one must be able to get descriptions of resources, properties, units, actions, enumeration values, etc. in a language applicable to the user.   Developer-centric descriptions (as might appear in a standard specification) and user-centric descriptions in another language can be quite different.   Although some devices might have the ability to locally serve out schema information in the languages appropriate to the markets they are sold in (e.g., those with a web server meant to be accessed via a browser often do), other devices might not have the storage capability to do so, even if they do have the capacity to store schema information.

Earlier we argued that devices should minimally be able to supply to a client at runtime the *location* of the schemas it implements, which may or may not be a location on the same device; we now extend that same concept to localized user descriptions.   A "description server" (which again may or may not be part of the same device) can supply the localized descriptions for one or more languages, and **devices**

should be able to supply to a client at runtime the location of the relevant localized description server for its schemas. In one implementation, a localized description server might also serve the machine-readable schema, with localized descriptions in it. In another implementation, a device itself might serve the machine-readable schema, but refer to another server on the Internet for getting localized descriptions. Other variations are of course possible.

Deep Mapping vs Algorithmic Mapping

We now return to the last, and perhaps most relevant to the WISHI workshop, use case listed above: allowing a gateway to algorithmically map resources from one ecosystem into "vendor-specific" (i.e., custom) resources of another.

Gateways for many resource types are created by a developer writing code to translate a resource (or set of resources) following a schema in one ecosystem, to a resource (or set of resources) following a different schema, and possibly also a different protocol, in another ecosystem. We refer to this as "deep mapping". As noted earlier, the schema relevant to a specific device may be only a subset of a standard schema, and so a gateway introspecting server devices can allow greater efficiency in deep mapping, and also allow accurate introspection by clients querying the gateway. Deep mapping, however, requires the developer to know a priori what the schemas are in each ecosystem, and hence is limited when new devices appear.

Algorithmic mapping, on the other hand, can use common code to translate arbitrary resources syntactically, while doing no semantic translation since it has no knowledge of semantics other than what appears in the machine-readable schema. Generally, the effect is to make a device appear in the client's protocol as a new vendor-specific (i.e., custom) type, rather than a standard type. Although the experience will not be as useful as if there were a deep mapping exposing the device as a standard type that the client expects, exposing a vendor-specific type can be much more useful than having no connectivity to the device at all.

We argue that **gateways should support deep mapping for the schemas they know about a priori, and algorithmic mapping for other resources/devices** for which the gateway has no deep mapping translator.

Algorithmic mapping requires introspection, as defined earlier, so that the algorithmic translation can preserve the semantics across the syntactic translation. It is also important to note that the virtual devices that the translator exposes should look to clients much like physical devices would, meaning that the client can introspect the algorithmically derived schemas exposed by the gateway.

OCF specifications, for example, define:

1) a framework for translation between arbitrary ecosystems that follows the above recommendations,
2) an algorithmic mapping between OCF and AllJoyn protocols (and discussions are ongoing regarding mapping between OCF and other technologies)
3) deep mapping between standard OCF and standard AllJoyn schemas (and work is ongoing regarding deep mapping between OCF and other ecosystems)