OPEN CONNECTIVITY FOUNDATION™

Derived Models for Interoperability Between IoT Ecosystems

J. Clarke Stevens, Piper Merriam

oneIoTa Overview

The Open Connectivity Foundation's (OCF) oneIoTa tool is essentially a web-based, Integrated Development Environment (IDE) for crowd-sourcing data models for the Internet of Things. This paper assumes a basic understanding of the features of oneIoTa and concentrates instead on the interoperability between IoT ecosystems supported by the OCF common data model and its derivative models.

Interoperability

One of the major problems with the Internet of Things is that there are many incompatible ecosystems. While some have addressed this problem by writing various types of converters between ecosystems, this becomes hard to scale. The OCF architecture and the oneIoTa tool use OCF data models as a "common" data model and a derived version of these models to define conversions between OCF and other IoT ecosystems. This makes all derived models interoperable with OCF and all other derived models through (at most) two conversion steps. As with all OCF data models derived models can be machine-read to automatically create code stubs.

Derived Models

Derived models use standard JSON schema syntax. Fundamentally, derived models provide a conversion mapping between OCF data models and similar data models in another IoT ecosystem. These conversions can be very simple (as in just a property name conversion) to extremely complex (as in converting between different numbers of properties with complex mathematics). Examples of the various conversions are described below.

Simple Mapping

Simple mapping just converts between different field names. Simple mapping accounts for field mappings between ecosystems. It can be used in combination with direct mappings as well as more complex mathematical conversions. In the example below, the derived model defines the field "lightness" to map to the OCF field "brightness." It also maps the derived ecosystem field "rgb_color" to the three OCF fields "red," "green," and "blue."

```
{
    properties: {
        lightness: {
            type: "number",
            oic_conversion: {
                oic_alias: "brightness",
            }
        },
        rgb_color: {
            type: "string",
            oic_conversion: {
                oic_alias: ["red", "blue", "green"],
            }
        }
    }
}
```

Simple Conversion

Simple conversion defines mathematical conversion between simple mappings. It does this using the two fields “from_oic” and “to_oic.” The mathematical conversion is defined in a string. The string is not validated. In this example, the field “darkness” in the derived model is converted to the field “brightness” in the OCF model by subtracting it from 255. A script that creates code stubs could read this model to identify input and output variables and use the conversion strings as comments that a coder could reference to properly implement the conversions in any programming language.

```
{
    properties: {
        darkness: {
            type: "number",
            oic_conversion: {
                to_oic: "brightness = 255 - darkness",
                from_oic: "darkness = 255 - brightness"
            }
        }
    }
}
```

The following example is a bit more complex and demonstrates the use of a list of strings. The derived model field “darkness_percentage” is converted to and from the OCF field brightness with two conversion steps for each direction.

```
{
    properties: {
        darkness_percentage: {
            type: "number",
            oic_conversion: {
                to_oic: [
                    "darkness = 255 * darkness_percentage",
                    "brightness = 255 - darkness"
                ],
                from_oic: [
```

```
                    "darkness = 255 - brightness",
                    "darkness_percentage = darkness_percentage / 255"
                ]
            }
        }
    }
}
```

Complex Conversion

Sometimes conversions are too complex to capture well in a from_oic/to_oic syntax. One example is conversion between RGB and HSV color schemes. This conversion requires intermediate variables and several conditional assignments. The example below shows how this could be done using the oic_conversions field. The from_properties and to_properties define the mappings while from_steps and to_steps define the list of conversion steps to be taken. Again, the strings defined in the steps are not validated, but are used as comments when code stubs are generated so the conversion process can be implemented in any programming language.

```
// OCF Model
{
    properties: {
        Hue: {
            type: "number",
        },
        Saturation: {
            type: "number",
        },
        Value: {
            type: "number",
        },
    }
}

// Derivative Model
{
    oic_conversions: [
        {
            from_properties: ["Hue", "Saturation", "Value"],
            to_properties: ["Red", "Green", "Blue"],
            from_oic_steps: [
                "C = V x S",
                "H_prime = H / 60 degrees",
                "X = C * (1 - |H_prime mod2 - 1|)",
                "if H is undefined: R, G, B = (0, 0, 0)",
                "if 0 < H_prime < 1: R_1, G_1, B_1 = (C, X, 0)",
                "if 1 < H_prime < 2: R_1, G_1, B_1 = (X, C, 0)",
                "if 2 < H_prime < 3: R_1, G_1, B_1 = (0, C, X)",
                "if 3 < H_prime < 4: R_1, G_1, B_1 = (0, X, C)",
                "if 4 < H_prime < 5: R_1, G_1, B_1 = (X, 0, C)",
                "if 5 < H_prime < 6: R_1, G_1, B_1 = (C, 0, X)",
                "m = V - C",
                "R, G, B = (R_1 + m, G_1 + m, B_1 + m)"
            ]
        }
```

```
    ],
    properties: {
        Red: {
            type: "number",
        },
        Green: {
            type: "number",
        },
        Blue: {
            type: "number",
        },
    }
}
```

One to Many

The following example shows how you can convert from one ecosystem with a single field to a derived model with multiple fields.

```
// OCF Model
{
    properties: {
        rgb: {
            type: "string",
        }
    }
}

// Derivative Model
{
    oic_conversions: {
        to_oic: [
            "rgb = r + b + g",
        ]
    },
    properties: {
        red: {
            type: "string",
            oic_conversions: {
                oic_alias: "rgb",
                from_oic: [...]
            }
        },
        blue: {
            type: "string",
            oic_conversions: {
                oic_alias: "rgb",
                from_oic: [...]
            }
        },
        green: {
            type: "string",
            oic_conversions: {
                oic_alias: "rgb",
                from_oic: [...]
            }
        },
```

```
    }
}
```

Many to One

This example shows how to convert between an OCF model with several fields to a derived model with a single field.

```
// OCF Model
{
    properties: {
        red: {
            type: "string",
        },
        blue: {
            type: "string",
        },
        green: {
            type: "string",
        },
    }
}

// Derivative Model
{
    properties: {
        rgb: {
            type: "string",
            oic_conversions: {
                oic_alias: ["red", "blue", "green"],
                from_oic: ["rgb = red + blue + green"],
                to_oic: [
                    "red = ...",
                    "blue = ...",
                    "green = ...",
                ]
            }
        }
    }
}
```

Derivative Model Validation

As with common “OCF” models, derived models are validated against JSON syntax. In addition, derived models are validated to ensure they are properly referenced to an OCF model. The derived model must include all the fields of the OCF model from which it is derived. These fields consist of oic_alias and from_properties within the oic_conversoins property.

This means that if a relevant model does not yet exist in OCF, it must be proposed and accepted by OCF before a derived model can be created in some other ecosystem. This expands the models within OCF and guarantees that the derived model with work with OCF models and all other derived models.

What's Next?

Derived models in OCF are an extremely powerful solution to interoperability between ecosystems in the Internet of Things. They are also very flexible in being able to support RESTful as well as other architectures. A prototype has been built showing interoperability between UPnP, AllSeen and Philips HUE light bulbs being adjusted in unison from a UPnP control point.

While oneIoTa is currently only populated with OCF models, derived UPnP models will soon be added. Other ecosystems interested in using oneIoTa and adding their models to the interoperable OCF ecosystem should contact OCF.