Developer support with code-generation

Dave Thaler, moderator

Several papers mentioned code generation

- Marcello Lioy, Dave Thaler, “Summary of AllSeen Alliance Work Relevant to Semantic Interoperability”
- Bryant Eastham, “Interoperability and the OpenDOF Project”
- Matthias Kovatsch, “Semantic Interoperability Requires Self-describing Interaction Models”
- Kerry Lynn, “Modeling RESTful APIs with JSON Hyper-Schema”
- …

Outline

- Use cases
- Data model requirements to enable codegen of basic functionality
- Additional things needed for some use cases
- Additional things useful for code readability/maintenance
- Various additional issues
- Code generator implementation decisions

Some code generation use cases

Given a formal data model:

1. Generate client-side library for talking to a Thing
2. Generate stubs for implementing a Thing
3. Generate tests for compliance
4. Generate fuzz tests
5. … others

Some requirements for data models / metadata to facilitate code generation (1/2)

- Generally required for basic functionality
  - Data model identifier
  - Data model version (whether explicit or implied)
  - Instance identifiers
  - Property/event/method names
  - Data types of their values (which might be links to other resources)
  - Allowed operations (create? read? update? delete? subscribe?)
  - Security requirements

Some requirements for data models / metadata to facilitate code generation (2/2)

- Needed for some use cases
  - Value constraints (e.g., ranges, relationships between values)
  - Validity lifetime of value (e.g., cachable? TTL? subscribe to actual value changes or just value-has-changed events?)
  - End-user description strings (e.g., label for each property, enum value, etc.)
  - Units
  - Default values
  - Display hints
  - Error messages
- Good for code readability/maintenance but not strictly required:
  - Named types
  - Developer comments

Various additional issues

1. How is instance discovery done, if multiple ways are allowed?
2. Strong type checking on client side?
3. Complex constraints (e.g., values that depend on other values)
4. How handle “optional” functionality on server side?
5. How handle “optional” functionality on client side – discovery time? capability negotiation? handle “not supported” failures?
   - Also includes issues of backwards compat & deprecated/obsolete func.
6. How to handle client side control loop, if any: open loop? closed loop? up to app?

Code generator implementation decisions

- Is code generation done at development time or runtime?
- Does code generator help retrieve data model or require it done a priori?
- Simplicity vs complexity, how let a simple app do simple things without precluding a complex app from doing complex things
- Code input formats:
  - Multiple data model languages or just one?
- Code output formats:
  - Multiple programming languages?
  - Multiple OS platforms?
  - Multiple serialization formats?
  - Multiple transport protocols?