Model-to-Model
Transforming with
ODL/plastic

September 26, 2019
Allan Clarke
Why are you here?

• **First public disclosure** of intent to upstream Plastic* into ODL
• Give an understanding of what this feature is
• Show kinds of problems it can help solve
• Want to get enthusiasm for adoption

* Brand new name (internal name was *Cartographer*)
What is the mapping problem?

What is a mapping problem?
- Occurs in internals of a system behind endpoints
- ODL context – moving from northbound to southbound representations
- Sometimes need to trivially convert data representation
  - JSON, XML, YML, other parse-able formats
- Sometimes need to change abstractions (1:1, N:1, N:N)
  - Morph one model into completely different model
  - Morph N models into one model
Available Solutions

What solutions are there today?
• Venerable XML/XSD/XSLT
• Apache Velocity
• DSLs
• Jolt
• Ad hoc programming
• Others
Everything has a tradeoff...

What are advantages and disadvantages?

• Can be very comfortable if you know underlying language
• Can be comforting to programmers
• Need to learn another language regardless of mapping complexity
• Imperative representation of conversions (hiding schemas in code)
Why ODL/plastic?

ODL/plastic Advantages

• Pay-as-you-go for complexity (field deployable changes)
• Declarative representations are emphasized (clear schemas)
• *Translation-by-intent* (say what you want, not how to do it)
• Deeper levels of abstraction to help keep custom logic schema-independence
• Can specify arbitrary morphing via plug-ins in JVM language
• Understands breaking large mapping problems up (both time and space) into small chunks
Justification

Solves problems like…
• Schema changes for device configurations across releases
• No more hard-wired dependency on vendor libraries
• In-the-field updating to support multiple versions of devices
• Light weight specifications avoid religiosity around “DRY"
Should you use ODL/plastic?

• Probably no, if your mapping problem rarely changes
• No, if you have abundant access to inexpensive programmer time
• No, if you don't care how hard it is to understand your translations
• No, if you are not particularly sensitive to regression breakages
• Probably no, if you have sub-millisecond throughput requirements
Plastic Mapping Specification

• Input schema
  • Exemplar-based specification (XML/JSON/…)
  • Variables defining important values

• Output schema
  • Exemplar-based specification (XML/JSON/…)
  • Variables defining bound values that are substituted

• Input payload
  • Same format of input schema
  • Partial match required against input schema

• Invoke
  • `Plastic.translate("ELN", "1.0", "JSON", "AA", "1.1", "XML")`
  • Schemas is an arbitrary file system hierarchy
Examples

Example-1: no coding - array expansion

Example-2: least schema-dependent coding using morpher “plug-in”

Example-3: highly dependent coding using classifier “plug-in”

Translation pipeline showing how all “plug-in”s relate to the flow
Example-2: schema-independent coding

- Via simplest possible “morpher” plug-in
- Shows manipulation AFTER variable binding
- Below is written in Java-style, not idiomatic Groovy

```java
class MyMorpher {
    def tweakValues(Map inputs, Map outputs) {
        outputs['ENDPOINT'] = inputs['ADDR-IN'] + ":" + inputs['PORT-IN']
    }
}
```
Example-3: **schema-dependent coding**

- Via simplest possible “classifier” plug-in
- Shows just-in-time schema name resolution
- Payload is parsed JSON (ie, arrays/maps)
- Below is written in Java-style, not idiomatic Groovy

```java
class MyClassifier extends SimpleClassifier {
    String classify(Object parsedPayload) // payload is parsed JSON {
        if (parsedPayload.astuff)
            return "schema-A"
        else
            return "schema-B"
    }
}
```
Plastic Pipeline

1. Payload parsed
2. Values bound
3. Output serializable
More Good Stuff

• Default values (in schema, passed in, or ad hoc logic)
• Multiple morphers
• Run-time reload of schemas and translation logic
• Multithreaded “batch” processing
• Simple synchronous API
• Command line runner for quick development
• Tutorial
Takeaways

• **General-purpose facility** usable in many contexts
• **Declaratively** manage schema changes
• Sometimes **no coding** required
• Insulate your code from changes
Open Questions

Thanks!

aclarke@luminanetworks.com