Lit
A domain model library for FHIR

Hugh Simpson
Staff Software Engineer

babylon

HL7® FHIR®
DevDays

Lit: A Domain Model Library for FHIR

Hugh Simpson

HL7 FHIR DevDays International 2022 | Hybrid Edition, Cleveland, OH | June 6–9, 2022 | @HL7 | @FirelyTeam | #fhirdevdays | www.devdays.com

ORGANIZED BY firely | HL7 International

WHY? Let’s build an observation with HAPI

Ceremony

```
new
HAPIObservation().setEffective(true)
```

=>

It does not compile:

```
type mismatch;
found   : Boolean(true)
required: org.hl7.fhir.r4.model.Type
       .setEffective(true)
```

Runtime exceptions

```
new HAPIObservation()
.setEffective(new BooleanType(true))
```

=>

It compiles - But …

```
java.lang.Error: Not the right type
for Observation.effective[x]:
boolean
```

Server says ‘no’

```
new HAPIObservation()
.setEffective(new HAPIDateTimeType(
  Date.from(ts.toInstant),
  TemporalPrecisionEnum.MILLI))
```

=>

We’re able to serialize the event and send it - but this is a bad thing. It is rejected when we run an integration test against the server.

WHAT could have happened?

Field validation at compile time

```
new HAPIObservation()
.setEffective(new HAPIDateTimeType(
  Date.from(ts.toInstant),
  TemporalPrecisionEnum.MILLI))
```

=>

Maybe we could have this not compile:

```
Unspecified value parameters code,
status.
```

?

Type validation at compile time

```
new HAPIObservation()
.setEffective(new BooleanType(true))
```

=>

Maybe we could have this not compile:

```
The type Boolean does not appear to
be a valid member of the union
FHIRDateTime \/ Period \/ Timing \/
ZonedDateTime
```

?

Do we need the wrapper types?

```
new BooleanType(true)
```

=>

Could we have something a bit more generic, such as

```
choice(true)
```

?

YES! Let's build an observation with Lit

Field validation at compile time?

```
LitObservation(
 effective = Some(choice(
   FHIRDateTime(dateTime = ts))))

=>
```

Success! It doesn't compile:

```
Unspecified value parameters code,
status.
      LitObservation(
                    ^
```

Type validation at compile time?

```
LitObservation(
 status = OBSERVATION_STATUS.FINAL,
 effective = Some(choice(true)),
 code = bmiCodeableConcept
)

=>

the type Boolean does not appear to
be a valid member of the union
com.babylonhealth.lit.hl7.UnionAlias
es.Union00107722725
 effective = Some(choice(true)),
                         ^
```

(Okay, this error message could be better)

Do we need the wrapper types?

Sort of! We've reduced the 50 wrapper types to one method. This feels like an improvement. Our final construction is:

```
LitObservation(
  status = OBSERVATION_STATUS.FINAL,
  effective = Some(choice(
    FHIRDateTime(dateTime = ts))),
  code = bmiCodeableConcept)
```

This is finally accepted by the compiler, and by the validation server

WHAT else might we want?

Subtypes

```
Observation(
  identifier = LitSeq(Identifier(system = Some("http://system.foo/bar"), value = Some("123"))),
  status = OBSERVATION_STATUS.FINAL,
  subject = Some(patientReference),
  category = LitSeq(vitalSignsCodeableConcept),
  value = Some(choice(someQuantity)),
  code = bmiCodeableConcept,
  effective = Some(choice(FHIRDateTime(dateTime = ts))),
  meta = Some(Meta(profile = LitSeq("http://hl7.org/fhir/StructureDefinition/bmi")))
)
```

Vs

```
Bmi(
  identifier = LitSeq(Identifier(system = Some("http://system.foo/bar"), value = Some("123"))),
  status = OBSERVATION_STATUS.FINAL,
  subject = patientReference,
  category = LitSeq(vitalSignsCodeableConcept),
  value = someQuantity,
  effective = choice(FHIRDateTime(dateTime = ts)),
  code = bmiCodeableConcept
)
```

Differences

- The ‘subject’ field is non-optional in Bmi observations
- The ‘value’ field is both non-optional and necessarily a ‘Quantity’ in Bmi observations
- The ‘effective’ field is both non-optional and restricted to one of ‘Period’ or ‘DateTime’ in Bmi observations
- The ‘meta.profile’ is automatically set for the Bmi class
- The ‘category’ field cannot be empty

```
...
category = LitSeq(),
...
```

```
found   : com.babylonhealth.lit.core.LitSeq[Nothing]
required:
com.babylonhealth.lit.core.NonEmptyLitSeq[com.babylonhealth.lit.core.model.CodeableConcept]
        category = LitSeq(),
                   ^
```

Compromises?

- Bmi class extends (via VitalSigns) the Observation class
- Field access cannot, consequently, return the same ‘unwrapped’ type as the constructor parameters – must return the same type for each field as in the first parent in which it’s declared

```
val value: Option[ValueChoice] = bmi.value
…
// Not all Unions have such unhelpful names. Unions of 2 or 3 types are named by types - e.g UnionDateOrPeriod
type ValueChoice = Choice[Union02118820890]
…

type Union02118820890 =
  Boolean \/ CodeableConcept \/ FHIRDateTime \/ Int \/ LocalTime \/
  Period \/ Quantity \/ Range \/ Ratio \/ SampledData \/ String
```

- Some accessors are provided

```
val value: Quantity = Bmi.extractValue(bmi)
```

WHAT else might we want?

Meta-information

```
val nonOptionalFieldNames: Seq[String] = Bmi.fieldsMeta
 .filterNot(_.tt.tag <:< LTT[Option[_]])
 .filter(f => f.tt.tag <:< LTT[NonEmptyLitSeq[_]] || !(f.tt.tag <:< LTT[LitSeq[_]]))
 .map(f => f.name + (if (f.isRef) "[x]" else ""))

      =>

List(status, subject, category, value[x], effective[x], code)
```

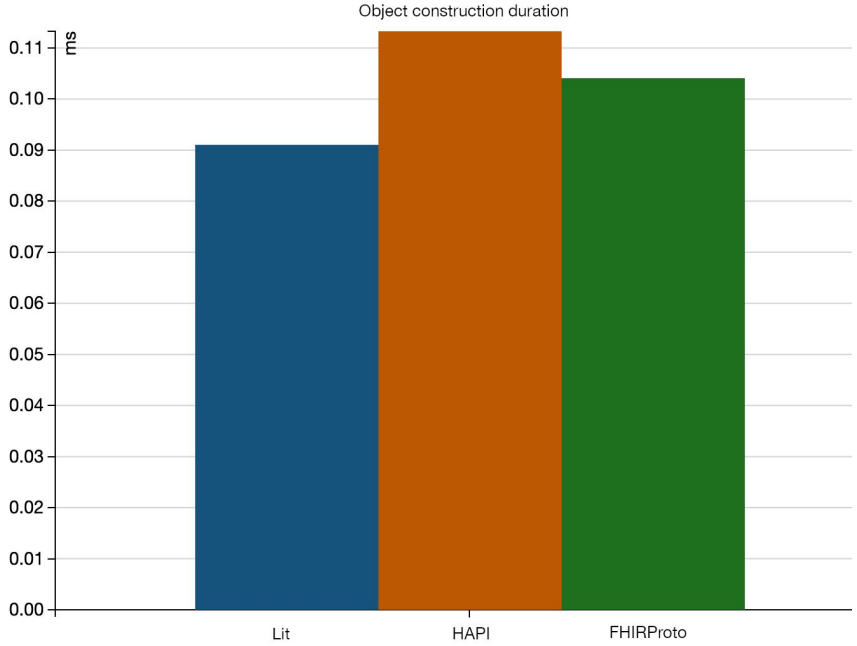
Transformations

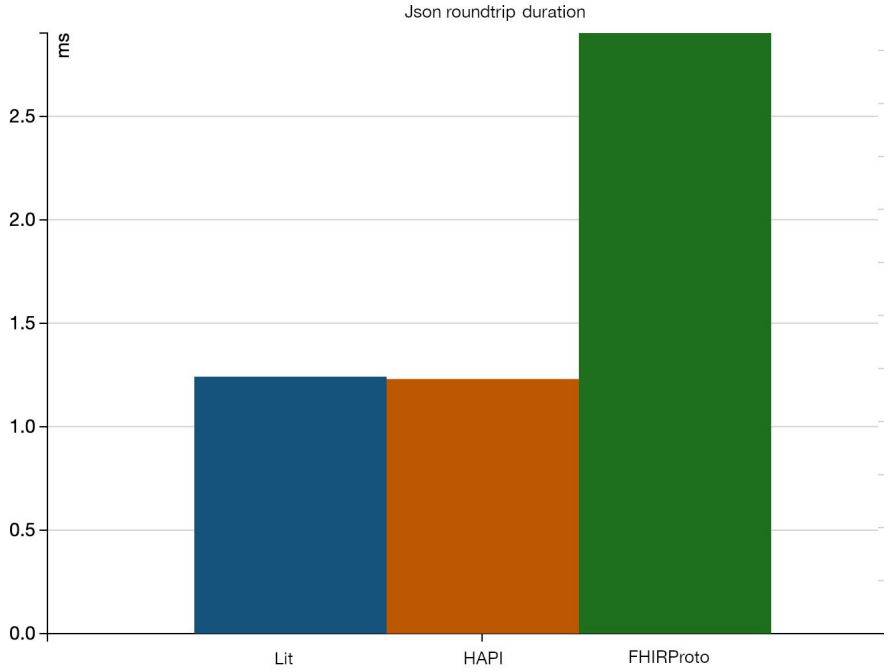
```
val mungedBundle: Bundle =
  bundle.nodalMap[Coding](classOf, _.updateCodeIfExists(c => if (c == "a-bad-code") "a-good-code" else c))
```

Extractions

```
val bundleCodings: LitSeq[Coding] = bundle.^^^[Coding]
```

At what cost?





WHAT if I don’t use Scala?

Java API

Builders

```
BmiBuilder.init(
  ObservationStatus.FINAL,
  patientReference,
  List.of(vitalSignsCodeableConcept),
  someQuantity,
  BmiBuilder.effective(new FHIRDateTime(ts, DateTimeSpecificity.Time)),
  bmiCodeableConcept
).withIdentifier(IdentifierBuilder.builder().withSystem("http://system.foo/bar").withValue("123")).build();
```

What’s different?

In Scala:

- Required parameters
- Default values where optional
- ‘choice’ method wrapper
- Advanced model introspection
- Syntax slightly eased by implicits and macros

In Java:

- Builder init methods
- Fluent DSL for optional params
- Named wrappers
- Some limitations to model introspection
- Lack of implicits and macros makes some interfaces clunkier

What’s the same?

- Shared empty/nonempty list type ‘LitSeq’
- Required params guaranteed to be present
- Immutable
- Sane inheritance
- Type refinements and hardcoded values reflected in subclass constructors
- New modules can be generated from internal or shared FHIR profiles

Gotchas?

- Startup time penalty
- Compile time penalty
- Reflection config must be provided to use with native-image

Did you know we're hiring?

Want to join a **global** organization?

Do you believe in Babylon's mission to
bring affordable and accessible healthcare
to every human on earth?

We're looking for
exceptional talent
across the technology organization.

Check out our Career Site to find your
next opportunity with Babylon.

babylonhealth.com/careers/technology

Thank you

babylon

Links

- https://github.com/babylonhealth/lit-fhir
- Further documentation