RAPID™ – Resource API Design Language
Pushing the limits of language usability with XText

TED EPSTEIN, FOUNDER AND CEO
TANYA FESENKO, LEAD DEVELOPER
EclipseCon NA 2015, Xtext Day
Overview

• RepreZen: solving API interoperability
• Challenge: a better API description language
• RAPID Language Features
  ◦ Optional Fluency: accommodating coding styles
  ◦ Keyword Sequence: encourage, don’t Enforce
  ◦ Indent-Based Block Scoping: life without curly braces

• Implementation Overview
• Summary
• Q&A
The Elevator Pitch

/ Unify Data Models

/ Federate APIs

/ Integrate Faster
• Frequently recommended, hard to implement
• Current reality: much easier to build from scratch than to reuse existing data models
• We want to change that!
The Elevator Pitch

Make Canonical Data Models Practical

- Frequently recommended, hard to implement
- Current reality: much easier to build from scratch than to reuse existing data models
- We want to change that!

Enable Highly Interoperable APIs

- Reusable data models: business-oriented, technology-independent, semantically rich
- Allow variability through realization
- Message payloads still conform to canonical model

Property Subsets

Metada

Contextual Constraints

Perspective

The Elevator Pitch

- Frequently recommended, hard to implement
- Current reality: much easier to build from scratch than to reuse existing data models
- We want to change that!

Enable Highly Interoperable APIs

- Reusable data models: business-oriented, technology-independent, semantically rich
- Allow variability through realization
- Message payloads still conform to canonical model

Property Subsets

Perspective

Metadata

Contextual Constraints

The Elevator Pitch

- Frequently recommended, hard to implement
- Current reality: much easier to build from scratch than to reuse existing data models
- We want to change that!

Enable Highly Interoperable APIs

- Reusable data models: business-oriented, technology-independent, semantically rich
- Allow variability through realization
- Message payloads still conform to canonical model

Property Subsets

Perspective

Metadata

Contextual Constraints
Make Canonical Data Models Practical

- Frequently recommended, hard to implement
- Current reality: much easier to build from scratch than to reuse existing data models
- We want to change that!

Enable Highly Interoperable APIs

- Reusable data models: business-oriented, technology-independent, semantically rich
- Allow *variability* through realization
- Message payloads still conform to canonical model

Significantly Lower Long-Term Integration Cost

- Service APIs speak the same language
- Clients can work with multiple APIs, without multiple DTOs, SDKs, etc.
- Stop runaway spending on remedial integration technology
Changing the way we build APIs

This needs to be built into the API modeling language.

Realization
- Business Domain Model
  - Subset
  - Perspective
  - Constraints
  - Metadata

Representation
- Resource Data Model
  - Media Type
  - Format
  - Conventions
  - Hyperlinks

Message Schema
Yet Another API Description Language?

YAADL? YAAPIDL?
There are at least four of these...

“The nice thing about standards is that you have so many to chose from.”
- Andrew Tanenbaum

... and they’re pretty good.
- Web-based editors
- API Documentation with integrated sandbox
- Syntax based on YAML or Markdown
- Interesting language-level reuse
- REST patterns built in

If we have to build an API modeling language, it has to be good.

... any ideas?
Who’s going to model these APIs?

Make it accessible, readable, versatile.

Developers! ...and Developers! Developers!

Business Analysts  Data Professionals

Subject Matter Experts  Enterprise Architects

What she said.
Demo – RAPID Usability Features

• **Indent Syntax:**
  ◦ Python-like block syntax, no curly braces
  ◦ Advanced highlighting to visualize structure

• **Order Assist:**
  ◦ Flexible sequence for declarations
  ◦ Reset to standard sequence using QuickFix or auto-format

• **Optional Fluency**
  ◦ Support terse and fluent syntax with optional keywords
  ◦ Semantic highlighting to distinguish optional vs. required keywords
Indent-Based Syntax

New Syntax –
indents define block scope

Previous Syntax –
braces for block scoping

Structural Highlighting -
editbox.sourceforge.net
Order Assist

objectResource TaxFilingObject type TaxFiling
referenceLink > taxpayer
targetResource PersonObject

**method PUT putTaxFiling**
request TaxFilingObject
response statusCode 200

**method GET getTaxFiling**
response TaxFilingObject statusCode 200
response statuscode 404

mediaTypes
application/xml

preferences

formatter

enable reordering of elements on formatting

preferences

warnings

configure project specific setting...

select the severity level for the following optional XtextDsl compiler problems:
- random order
  - collectionResource: warning
  - objectResource: warning

**method PUT putTaxFiling**
request TaxFilingObject
response statusCode 200

**method GET getTaxFiling**
response TaxFilingObject statusCode 200
response statuscode 404
Optional Fluency

Fluent

```plaintext
objectResource FluentSyntax type Person
  with all properties including otherNames!
```

Terse

```plaintext
objectResource TerseSyntax type Person
  all properties otherNames!
```
Implementation Walkthrough
Work with Xtext

Xtext
Grammar
(*.xtext file)
Work with Xtext

Xtext Grammar (*.xtext file) → Generated code (Parser and UI)
Work with Xtext

The rest
(customized from generated code)

Lexer
Formatter
Content assist
Code Templates
Prefs
Cross-reference Scope
Code highlighter
Validation/Quickfix
Outline
...

Xtext Grammar (*.xtext file)

Generated code (Parser and UI)
Three features

for (each feature):

Description
Grammar
Code Customization
Feature 1: Automate canonical ordering
Order Assist

```plaintext
objectResource TaxFilingObject type TaxFiling
referenceLink > taxpayer
targetResource PersonObject

method PUT putTaxFiling
request TaxFilingObject:
response status code 200

mediaTypes
- application/xml

method GET getTaxFiling
response TaxFilingObject status code 200
response status code 404
```

Methods:

```plaintext
objectResource TaxFilingObject type TaxFiling
referenceLink > taxpayer
targetResource PersonObject

method PUT putTaxFiling
request TaxFilingObject:
response status code 200

mediaTypes
- application/xml

method GET getTaxFiling
response TaxFilingObject status code 200
response status code 404
```
Implementation: Grammar

The rest
(customized from generated code)

Xtext Grammar (*.xtext file) → Generated code
Allow arbitrary order in grammar

```
(  
singleValuedProperty = Rule1 | multiValuedProperty += Rule2

*)
```
Allow arbitrary order in grammar

```
( singleValuedProperty = Rule1 | multiValuedProperty += Rule2 )
```
Allow arbitrary order in grammar

(c

singleValuedProperty = Rule1 | multiValuedProperty += Rule2
)

*
Sort List< INode>

Sorting rule:
Sequence<GrammarElement>

Grammar Tree

Compare INodes:
• Using sorting rules
• Preserving:
  • List delimiters
  • Whitespaces
  • Attached symbols (*, +)
  • Order of elements of the same kind
Implementation: Grammar

The rest
(customized from generated code)
Validation

MyJavaValidator#
checkMyElement(SemanticElement)
Validation

MyJavaValidator #
checkMyElement(SemanticElement)

Semantic Element -> Syntax Tree Node -> getIncorrectly OrderedChildren() -> Incorrectly ordered Nodes

NodeModelUtils::getNode()
Implementation: Grammar

The rest
(customized from generated code)

Xtext Grammar (*.xtext file) ➔ Generated code

Formatter
Formatter

DefaultNodeModelFormatter#format():

```
DefaultNodeModelFormatter

NodeModelStreamer

feedTokenStream(rootNode)

getLeafs() : INode*

loop
  [for each parse tree node]
  writeSemantic() / writeHidden()

ITextRegion

DefaultNodeModelFormatter

NodeModelStreamer
```
Formatter

DefaultNodeModelFormatter#format():

```
DefaultNodeModelFormatter

NodeModelStreamer

feedTokenStream(rootNode)

getLeafs(): INode

loop [for each parse tree node]

writeSemantic() / writeHidden()

ITextRegion

DefaultNodeModelFormatter

NodeModelStreamer
```
Formatter

DefaultNodeModelFormatter#format():

INode:
- offset
- text
- length
- grammarElement
- semanticElement
Formatter

DefaultNodeModelFormatter#format():

- `feedTokenStream(rootNode)`
- `getLeafs() : INode`*
- Loop: `[for each parse tree node]`
- `writeSemantic() / writeHidden()`
- `ITextRegion`

Writes tokens to character stream and then flushes it.

The write methods create formatted LineEntries with use of formatter configuration (ElementLocators).
Formatter
DefaultNodeModelFormatter#format():

TextRegion – offset and length in the char stream populated in the write methods
Formatter

DefaultNodeModelFormatter#format():

Place for customization
Implementation: Grammar

The rest
(customized from generated code)

Xtext Grammar (*.xtext file) → Preferences page

Generated code
Usability: Preferences
Testing

5! = 120

How many permutations for five elements?
Testing
Xtend Active Annotations

```java
@Active(typeOf(ArbitraryOrderFormatterParticipant))
annotation ArbitraryOrderFormatter {
    ...
}

class ArbitraryOrderFormatterParticipant

@ArbitraryOrderFormatter
class ArbitraryOrderFormatterTest_wIncl {
    def String textualModel() {
        "..."
    }
    val String includedProperties = "..."
    val String referenceEmbed = "..."
    val String method1 = "..."
    val String method2 = "..."
    val String explicitMediaTypes = "..."

    ...
}
```

120 green tests!
Changes: Automate Canonical Ordering

The rest (customized from generated code)

- Validation
- Formatter
- Preferences page

Enable arbitrary order

Xtext Grammar (*.xtex file) ➔ Generated code
Changes: Automate Canonical Ordering

- Errors for single-value property override
- Warnings for "incorrect" order

The rest (customized from generated code)

Validation

Formatter

Preferences page

Xtext Grammar (*.xtext file)

Generated code
Changes: Automate Canonical Ordering

Generated code

Validation

Formatter

Preferences page

Xtext Grammar (*.xtext file)

Restore canonical order

Changes: Automate Canonical Ordering
Changes: Automate Canonical Ordering

The rest
(customized from generated code)

Validation

Formatter

Preferences page

enable/disable validation, reordering in formatter

Xtext Grammar (*.xttext file)

Generated code
Feature 2: Optional fluency
What is Optional Fluency?

Fluent:
```
objectResource FluentSyntax type Person
  with all properties including otherNames!
```

Terse:
```
objectResource TerseSyntax type Person
  all properties otherNames!
```
Implementation: Grammar

The rest
(customized from generated code)

Xtext Grammar (*.xtext file) → Generated code
Implementation
Implementation

objectResource FluentSyntax type Person
with all properties including
otherNames!

(with 'all' 'properties'
(including
   // included properties
)
(excluding
   // excluded properties
)
)

objectResource TerseSyntax type Person
all properties
otherNames!
Changes: Optional Fluency

The rest
(customized from generated code)

Optional fluency

Xtext Grammar (*.xtext file)

Generated code

No customization
Feature 3: Indent-based block scoping in Xtext
Indent-based block scoping in Xtext

With curly braces

```objectResource TaxFilingObject type TaxFiling {
  URI taxFilings/{id} {
    /** filingID of the requested TaxFiling */
    required templateParam id property filingID
  }
  referenceLink > taxpayer {
    targetResource PersonObject
    targetProperties {
      taxpayerID
    }
  }
} mediaTypes application/xml,
method GET getTaxFiling {
  response TaxFilingObject statusCode 200
},
```

Python-like notation

```objectResource TaxFilingObject type TaxFiling
  URI taxFilings/{id} {
    /** filingID of the requested TaxFiling */
    required templateParam id property filingID
  }
  referenceLink > taxpayer
  targetResource PersonObject
  targetProperties
    taxpayerID
} mediaTypes application/xml
method GET getTaxFiling
  response TaxFilingObject statusCode 200
  response statusCode 404```
Indent-based block scoping in Xtext
Implementation: Grammar

The rest
(customized from generated code)

Xtext Grammar (*.xtex file) → Generated code
Grammar

```
terminal NL   : (\r?\n)+ \t*;
terminal WS    : ( ' ' | \t)+;
terminal BEGIN : 'r';
terminal END    : 'n';
```

NL - newline
WS – whitespace
BEGIN – indent
END - outdent
Grammar

hidden(WS, ML_COMMENT, ML_COMMENT_NL, SL_COMMENT)

terminal NL : ('\r'? '\n')+ '\t'*;
terminal WS : (' '| '\t')*;

terminal BEGIN : '\r';
terminal END : '\n';

NL - newline
WS – whitespace
BEGIN – indent
END - outdent
Grammar

hidden(WS, ML_COMMENT, ML_COMMENT_NL, SL_COMMENT)

terminal NL : ('\r'?\n')+ '\t'�;
terminal WS : (' '|\t')+;
terminal BEGIN : 'r';
terminal END : 'n';

DataModel returns DataModel:
  {DataModel}
  (documentation=Documentation)?
  NL+ 'dataModel'
  name=ID
  BEGIN
  (ownedDataTypes+=DataType)*
  NL? END
  ;
Implementation: Grammar

Xtext Grammar (*.xtext file) → Lexer → The rest (customized from generated code) → Generated code
Custom Lexer

Generated / Xtext code

Custom code

Indent-based Syntax in Xtext - tinyurl.com/pgcowct

Insert BEGIN and END token
Implementation: Grammar

The rest
(customized from generated code)

Content Assist

Xtext Grammar (*.xtext file) → Generated code
Custom Lexers

Generated / Xtext code

Custom code

Plural now

Content-assist needs its own lexer
Custom Lexers

Generated / Xtext code

Custom code
Content Assist

Also see “Enhancing Content Assist” from XtextCON 2015 - tinyurl.com/onou2hu
Content Assist

Also see “Enhancing Content Assist” from XtextCON 2015 - tinyurl.com/onou2hu
Content Assist

Also see “Enhancing Content Assist” from XtextCON 2015 - tinyurl.com/onou2hu
Content Assist

Place for customization
Content-Assist Context Factory

Initialize data

Analyze data to create content assist contexts

Content Assist Contexts
Content-Assist Context Factory

Syntax tree nodes – lastCompleteNode, lastVisibleNode, currentNode, datatypeNode
Semantic model - current model

Initialize data → Analyze data to create content assist contexts → Content Assist Contexts
Content-Assist Context Factory

- Initialize data
- Analyze data to create content assist contexts
- Start a new token, complete current one, collect available alternatives, etc…

Content Assist Contexts
Content-Assist Context Factory

- Initialize data
- Analyze data to create content assists contexts
- Customize to respect indents

Content Assist Contexts
Implementation: Grammar

The rest
(customized from generated code)

Formatter

Xtext Grammar (*.xtext file) → Generated code
Formatter

```java
for (Pair<RuleCall, RuleCall> pair :
    findRulePairs(f.getGrammar(), f.getBEGINRule(), f.getENDRule())) {
    c.setIndentation(pair.getFirst(), pair.getSecond());
}
```

// disable line wrapping for all grammar rules
```java
for (AbstractRule rule : GrammarUtil.allRules(f.getGrammar())) {
    c.setNoLinewrap().around(rule);
    for (Keyword keyword : GrammarUtil.containedKeywords(rule)) {
        c.setNoLinewrap().around(keyword);
    }
}
```
```java
c.setNoSpace().around(f.getBEGINRule());
```

Assign correct value to FormattingConfigBasedStream

```java
#indentationLevel
```

No line wraps should be introduced by formatter – line wraps change structure of Python-like models

No spaces around “invisible” rules – synthetic (BEGIN and END) and NL (newline)
Changes: Indent-based Syntax

The rest
(customized from generated code)

Xtext Grammar (*.xtxt file)

Generated code

Add BEGIN and END terminal rules

Lexer

Content Assist

Formatter
Changes: Indent-based Syntax

- Synthesize BEGIN and END tokens
- The rest (extracted from generated code)

- Lexer
- Content Assist
- Formatter

Xtext Grammar (*.xtext file) → Generated code
Changes: Indent-based Syntax

Xtext Grammar (*.xtext file)

Generated code

Lexer

Content Assist

Formatter

Show proposals according to indent level

(customized from generated code)
Changes: Indent-based Syntax

The rest (customized from generated code)

- Lexer
- Content Assist
- Formatter

Preserve NLs and indents

Xtext Grammar (*.xtext file) → Generated code
Summary

• Still an ongoing experiment
  ◦ Early feedback from users has been very positive.
  ◦ We welcome your feedback!

• API Studio is a commercial product, still pre-launch

• Offering a free license with 1-year maintenance to Conference attendees (1 per organization)

• Register at get.reprezen.com with invitation code eclipsecon2015

• Contacts:
  ◦ ted.epstein@reprezen.com
  ◦ tatiana.fesenko@reprezen.com
Questions

THANK YOU!

RepreZen API Studio