Schematron Based Semantic Constraints Specification Framework & Validation Rules Engine for JSON

Advisor: Dr. Lixin Tao

Student: Dr. Amer Ali

DPS 2014

Seidenberg
School of Computer Science and Information Systems
PACE UNIVERSITY
Work toward greatness.

Abstract

- JavaScript Object Notation (JSON) has emerged as a popular format for business data exchange. It has a **grammar-based** schema language called – **JSON Schema** (IETF draft 7). The JSON Schema provides facilities to specify **syntax constraints** on the JSON data. There are a number of **tools available** in a variety of programming languages for JSON Schema validation. However, JSON does not have a standard or a framework to specify the **semantic constraints**, neither it has any **reusable validation tool** for semantic rules. In order for JSON data validation to be effective, it needs both syntax and semantic specification standards/frameworks and validation toolset[2].
- **XML** is another popular format for business data exchange that preceded JSON. XML has a mature ecosystem for specifying and validating syntax and semantic constraints. It has XML Schema and several other syntax constraints specification standards. It has Schematron as a semantic constraints specification language which is an ISO standard [ISO/IEC 19757-3].
- This study **proposes** a **framework** for specifying **semantic constraints** for JSON data in JSON format, drawing upon the power, simplicity, and semantics of Schematron standard. A **reusable JavaScript/NodeJS based validation tool** was also developed to process the JSON semantic rules.
- The framework **assumes** that due to inherent differences between **XML and JSON** data formats, not all Schematron concepts will be applicable to this study.

Why Business Data Validation?

- **$ 1 billion** Automotive Industry losses
  - National Institute of Standards and Technology (NIST) study[9]
- **10-25% of total revenue** losses for an org
  - Larry English [4]
- **40%** initiatives fail due to invalid data
  - Gartner 2011 report [11]
- **26 – 32 %** bad data in orgs
  - Experian 2015 study [12]
- **$3.1 trillion** estimated total cost
  - of bad data to the US economy [1]
  - Tibbett -based on $314B Healthcare industry[10]

Causes of Data Quality Issues

- Singh et al[13] 2010 study
- degrades during data handling stages
  - at the **source**
  - during **integration**/profiling
  - during data **ETL** (extraction, transformation and loading)
  - even data **modeling**

When to Validate Data ?

- The SiriusDecisions **1-10-100 Rule**
- W. Edwards Deming [14]

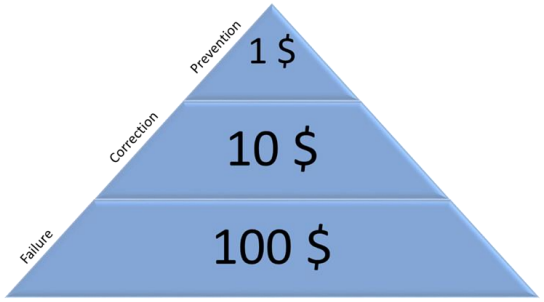
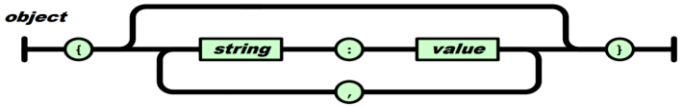


Figure 1 1-10-100 Rule

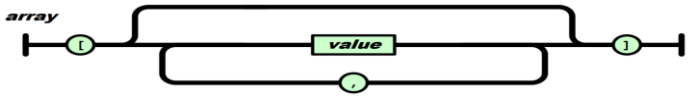
JSON – JavaScript Object Notation

- JSON (JavaScript Object Notation) is a:
- Lightweight,
- text-based,
- language-independent data interchange format
- **Based on a subset of the JavaScript**, ECMA-262
- Officially name “The JSON Data Interchange Format”
  - Ecma Standard in 2013 (**ECMA 404**)
- Looks like **data strucures used in many languages**

- Two main structures
  - **Object**: **Collection** of name/value pairs
    - Object, record, struct, dictionary, hashtable, keyed-list
    - *{ “key1”: value, “key2”: value2}*



  - **Array**: An **ordered list** of values
    - Array, vector, list or sequence
    - *[value1, value2, valueN]*



  - **Value**: object, array, number, string, true, false, null

```
{
  "doc": {
    "prologue": {
      "title": "Faster than light travel",
      "subtitle": "From fantasy to reality",
      "author": [
        {
          "member": "yes",
          "email": "cemereuwa@nasa.gov",
          "name": "Chikezie Emereuwa"
        },
        {
          "member": "yes",
          "email": "okey.agu@navy.mil",
          "name": "Okechukwu Agu"
        }
      ]
    },
    "section":{}
  }
}
```

Listing 1

Loan Data Example

XML

```
<loan_data>
<loans>
      <loan>
         <loan type="FHA">
         <loan_id> 989773 </loan_id>
         <customer_id>FLN498765</customer_id>
         <data_time>20100601120000</data_time>
         <amount>250000 </amount>
         <interest_rate> 3.75 </interest_rate>
         <prime_rate> 3.25 </prime_rate>
         <mip_rate> 1.5 </mip_rate>
         <down_payment> 5</down_payment>
         <loan_restricted/>
         <escrow>true</escrow>
         <origination_id> branch </origination_id>
         <branch_id>34567</branch_id>
         <electronic>true</electronic>
         <email>john.doe@gmail.com</email>
         <customer>
                  <customer_id > JD689457 </customer_id>
                  <customer_fname>John </customer_fname>
                  <customer_lname>Doe </customer_lname>
                  <customer_address> 4 Way Loop, New York, NY 10038
                  </customer_address>
          </customer>
        <loan>
</loans>
</loan_data>
```

Listing 2

JSON

```
{
"loan_data":{
    "loans":[
                {
                        "loan_id":"1234567",
                        "loan_type":"FHA",
                        "customer_id":"JD689457",
                        "data_time":"20100601120000",
                        "amount":500000,
                        "interest_rate":3.75,
                        "prime_rate":3.25,
                        "mip_rate":1.5,
                        "down_payment":5,
                        "loan_restricted":false,
                        "escrow":true,
                        "origination_id":"branch",
                        "branch_id":"5463",
                        "electronic":true,
                        "email":"john.doe@gmail.com",
                        "customer":{
                                "customer_id":"JD689457",
                                "customer_fname":"John",
                                "customer_lname":"Doe",
                                "customer_address":" 4 Way Loop,
                                New York, NY 10038"
                                 }
                    }
                ] }
}
```

Listing 3

Data Validation (Analogy)

- Semantic
  - **Co-constraints**
    - class = business (20lbs)
    - class = economy (14lbs)
- Syntax
  - **Structure** of data
    - H=56 cm W=45 cm D=25 cm
- Specifications
  - **Schema**
  - Standard
  - Framework
- Validators
  - **Processor**

Figure 2

Figure 3

Figure 4

JSON Constraint Specification & Validation

- Syntax
  - Specficication
    - **JSON Schema**
      - **IETF Draft**
  - Validation Tools
    - **Multiple**
- Semantic
  - Specification
    - **None**
  - Validation Tools
    - **None standard**
    - **Host platform**

Figure 5

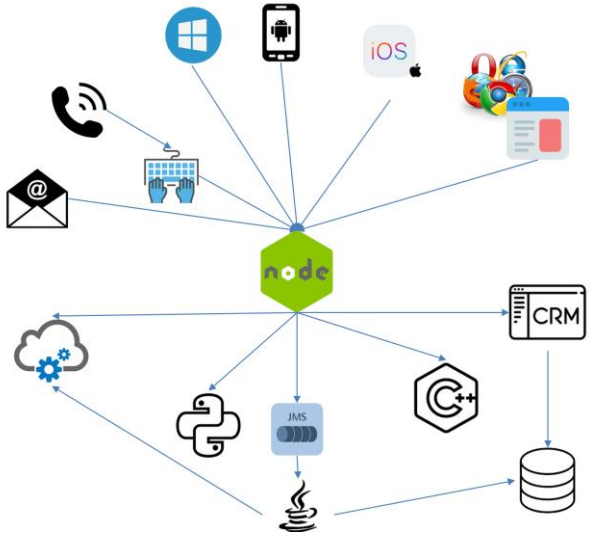


Figure 6

Syntax Validation

- Loan type should be **present**
- Loan type should be one of the values: FHA, Traditional, Jumbo, Commercial
  - **Enum**
- Loan id should be present
  - Loan id should be **minimum** 7 chars and **maximum** 8 chars
- Customer id should be present
- Amount should be present
- Amount should be minimum 100,000 [minimum = 100000]
- Interest rate should be present
  - **Default** interest rate is 3.5%
- Prime rate should be present
- Mip rate is **optional**/conditional
  - Min .85%, max 1.75%
- Down payment should be present
- Escrow should be present
- Origination id is required
- Origination id should be one of: branch, web, phone, third party
- Branch id is optional/conditional
- If **electronic = true**, valid email should be present
  - Dependencies : electronic ["customer_email"]
  - Email: "**format**": email
- Customer_name is required

JSON Schema

```
"required": [
  "loan_id",
  "loan_type",
  "customer_id",

"loan_type": {
  "type": "string",
  "enum": ["FHA", "Traditional", "Jumbo", "Commercial"]

  "loan_id": {
    "type": "string",
    "minLength":7,
    "maxLength":8
  },

"amount": {
  "type": "number",
  "multipleOf": 1,
  "minimum": 100000,
  "exclusiveMinimum": false
},

"interest_rate": {
  "type": "number",
  "default":3.5
},

"mip_rate": {
  "type": "number",
  "maximum": 1.75,
  "minimum": 0.85,
  "exclusiveMaximum": false,
  "exclusiveMinimum": false

},
"dependencies": {
  "electronic" :["email"],
  "credit_card": ["billing_address"],
  "billing_address": ["credit_card"]
```

Listing 5

Semantic Validation

- If loan type is **FHA**, amount can't **exceed** 500K
- If loan type is FHA, **mip_rate can't be 0** or less
- If loan type is **traditional**, amount can't **exceed 1MM**
- If loan type is **jumbo**, the amount can't be **less than 1M**
- **Interest rate** should at least be **.25 %** more than **prime** rate
- If loan type is not FHA, **down payment** can't be less than **20**%
- If origination id is '**branch**' then '**branch_id**' should be present
- Customer **id** under loan and customer id under customer should match

```
{
"loan_data":{
"loans":[
 {
        "loan_id":"1234567",
        "loan_type":"FHA",
        "customer_id":"JD689457",
        "data_time":"20100601120000",
        "amount":500000,
        "interest_rate":3.75,
        "prime_rate":3.25,
        "mip_rate":1.5,
        "down_payment":5,
        "loan_restricted":false,
        "escrow":true,
        "origination_id":"branch",
        "branch_id":"5463",
        "electronic":true,
        "email":"john.doe@gmail.com",
        "customer":{
                "customer_id":"JD689457",
                "customer_fname":"John",
                "customer_lname":"Doe",
                "customer_address":" 4 Way Loop, New York, NY 10038"
    }
} ]}
```

Listing 6

Limitations of Current JSON Validation

- JSON Schema has very limited **semantic facilities**
- No semantic constraints standard/**framework**
- No platform **agnostic** tools
  - host platform only
- No **progressive** validation
  - mechanism to divide the validation into phases to support validation of a particular constraint or workflow
- No **dynamic** validation
  - assume that all constraints are of equal severity and
  - must be treated the same way at the same time.
  - No mechanism to invoke a subset of constraints based on the needs.
- No **logical groupings** of constraints
  - don't support logical grouping of constraints based on various needs outside their structural formations
- Not able to handle **variance** in the schema
  - No facility on consumer side to handle variance
- No **abstractions** higher than elements
  - Simple and complex elements only
- No facility to define **business rules**
  - Heavily oriented to tech developers
  - No facility for BA, QA, Legal, and Compliance people
- No facility to specify constraints on **graph/tree pattern** relationships
  - Any addressable location for any other addressable location
- Assertion **messages** not human readable
  - Technical stack traces only
- Lack of **efficiency**
  - Select a single node and then test all assertions against it

XML

- Extensible Markup Language (**XML**) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.
- **Syntax Validation**
  - XML Schema, DTD, RELAXNG
- **Semantic Validation**
  - Schematron
- **Multiple validators for both**

XML Instance

```
<address>
        ...
        <city> New York City </city>
        <state> NY </state>
        <zipcode> 10038 </zipcode>
        ....
</address>
```

XML Syntax Constraints (XML Schema)

```
<xs1:schema xmlns:xs1="http://www.w3.org/2001/XMLSchema">
    <xs1:element name="address">
        <xs1:complexType>
            <xs1:sequence>
                <xs1:element name="city" type="xs1:string"/>
                <xs1:element name="state" type="xs1:string"/>
                <xs1:element name="zipcode" type="xs1:string"/>
            </xs1:sequence>
        </xs1:complexType>
    </xs1:element>
</xs1:schema>
```

XML Semantic Constraints (Schematron)

```
<rule context="address">
        <assert test="city">Address must have city name </assert>
        <assert test="state">Address must have state name</assert>
        <assert report ="zipcode">Address has a zipcode </report>
</rule>
```

Listing 7

Schematron

- **Schematron is a rule-based XML validation schema language for making assertions about the presence or absence of patterns in XML trees**

- **Capable of specifying rules that syntax based schema languages can't**
  - Control the contents of an element vial its siblings

- **Fundamental difference**
  - Syntax-based: **grammar based**
  - Schematron: based on **finding tree patterns**

- **Rick Jelliffe invented it at Academia Sinica, Taipei (1999-2001)**
  - "a feather duster to reach the corners that other schema languages cannot reach"

- **Standardized by the ISO as:**
  - "Information technology, Document Schema Definition Languages (DSDL), Part 3: Rule-based validation, Schematron (**ISO/IEC 19757-3:2016**)"

```
schema
    title
    phase
    pattern+
        rule+
            (assert or report)+
```

- **Main building blocks**
  - **Schema:** Top level element.Everything enclosed in it. Attributes – title, schemaVersion, queryBinding and defaultPhase
  - **Phase:** Abstraction. Specifies a group of patterns to be activated. #DEFAULT and #ALL special phases
  - **Pattern:** Abstraction. Set of rules elments. Not same as regex pattern.
  - **Rule:** One or more assertions applied to 'context' nodeset selected via query language
  - **Context:** Query language expression to select nodeset
  - **Assertions:** Contains 'test'. Tests are conditions that are applied to context. A 'message' is displayed. Assert vs. Report
  - **Reporting:** Validation result report. Left up to implementations

Schematron Data Model

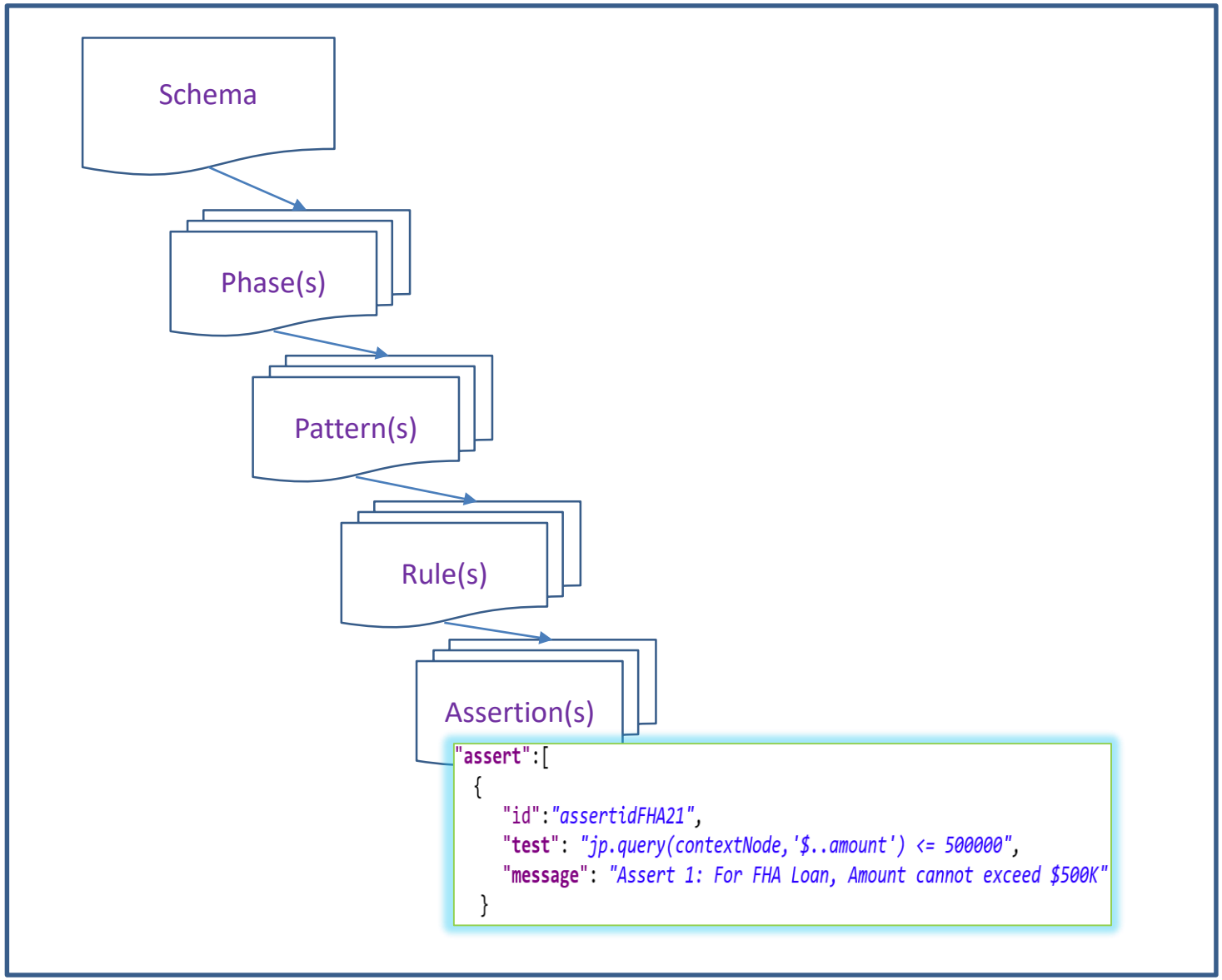


Figure 7

Solution Methodology

- ISO **Schematron** 19757-3 as base co-constrain/validation rules specification standard
- **JSON** as rules specification **data format**
- **JSONPath** as query language
- **JavaScript** as implementation language
- Input-Process-Output (**IPO**) as software implementation pattern
- **Node.js** as runtime platform
- **API Led** Connectivity / Microservice as architecture
- **Eclipse** as Integrated Development Environment (IDE)
- **GitHub** as repository
- Node Package Manager (**NPM**) as registry

JSON Schematron Rules

```
{"schema":{

  "id":"Loan Data Rules",
  "title":"Schematron Semantic Validation Rules",
  "schemaVersion":"ISO Schematron 2016",
  "queryBinding":"jsonpath",
  "defaultPhase":"phaseid1",
      "phase":[
         {
         "id":"phaseid1",
         "active":["patternid1"]
         }],
             "pattern":[
                 {
                 "id":"patternid1",
                 "title":"Loan Amount Pattern",
                 "rule":[
                     {
                      "id":"FHArule1",
                     "context": "$.loan_data.loans[?(@.loan_type === 'FHA')]",
                     "assert":[
                       {
                          "id":"assertidFHA21",
                          "test": "jp.query(contextNode,'$..amount') <= 500000",
                          "message": "Assert 1: For FHA Loan, Amount cannot exceed $500K"
                        }
     ]}]}]}}
```

Listing 8

Semantic Validation

- If loan type is **FHA**, amount can't **exceed** 500K
- If loan type is FHA, **mip_rate can't be 0** or less
- If loan type is **traditional**, amount can't **exceed 1MM**
- If loan type is **jumbo**, the amount can't be **less than 1M**
- **Interest rate** should at least be **.25 %** more than **prime rate**
- If loan type is not FHA, **down payment** can't be less than 20%
- If origination id is **'branch**' then **'branch_id**' should be present
- **Customer id** under loan and customer id under customer should match

New Rules

```
{
    "id":"rule22",
    "abstract":false,
    "context": "$.loan_data.loans[?(@.loan_type === 'FHA')]",
    "assert":[
      {
        "id":"assertid221",
        "test": "jp.query(contextNode,'$..amount') <= 500000",
        "message": "Assert 221: For FHA Loan, Amount cannot exceed $500K"
      },
      {
        "id":"assertid222",
        "test": "jp.query(contextNode,'$..mip_rate') > 0",
        "message": "Assert 222: For FHA loans, You must have MIP (Mortgage Insurance Premium)"
      }

    "context": "$.loan_data.loans[?(@.loan_type === 'Traditional')]",
    "assert":[
      {
        "id":"assertid31",
        "test": "jp.query(contextNode,'$..amount') <= 1000000",
        "message": "Assert 31: For Traditional Loan, Amount cannot exceed $1MM"
      }

     "context": "$.loan_data.loans[?(@.loan_type === 'Jumbo')]",
     "assert":[
       {
         "id":"assertid41",
         "test": "jp.query(contextNode,'$..amount') >= 1000000",
         "message": "Assert 41: For Jumbo Loan, Amount cannot be less than $1MM"
       }

    "context": "$.loan_data.loans.*",
    "assert":[
      {
        "id":"assertid81",
        "test": "(jp.query(contextNode,'$..interest_rate') - jp.query(contextNode,'$..prime_rate')) >= .25",
        "message": "Assert 81: Interest Rate should be atleast .25 points more than Prime Rate"
      }

    "context": "$.loan_data.loans[?(@.loan_type != 'FHA')]",
    "assert":[
      {
        "id":"assertid251",
        "test": "jp.query(contextNode,'$..down_payment') >= 20",
        "message": "Assert 251: For non-FHA loans, Minimum 20% downpayment is required"
      }

    "context": "$.loan_data.loans[?(@.origination_id === 'branch')]",
    "assert":[
      {
        "id":"assertid261",
        "test": "jp.query(contextNode,'$..branch_id') != ''",
        "message": "Assert 261: Missing Branch ID"
      }

  "context": "$.loan_data.loans.*",
  "assert":[
    {
      "id":"assertid271",
      "test": "jp.query(contextNode,'$[?(@.customer_id == @.customer.customer_id)]') != false",
      "message": "Assert 271: Customer ID mismatch"
    }
```

Listing 9

API Layers

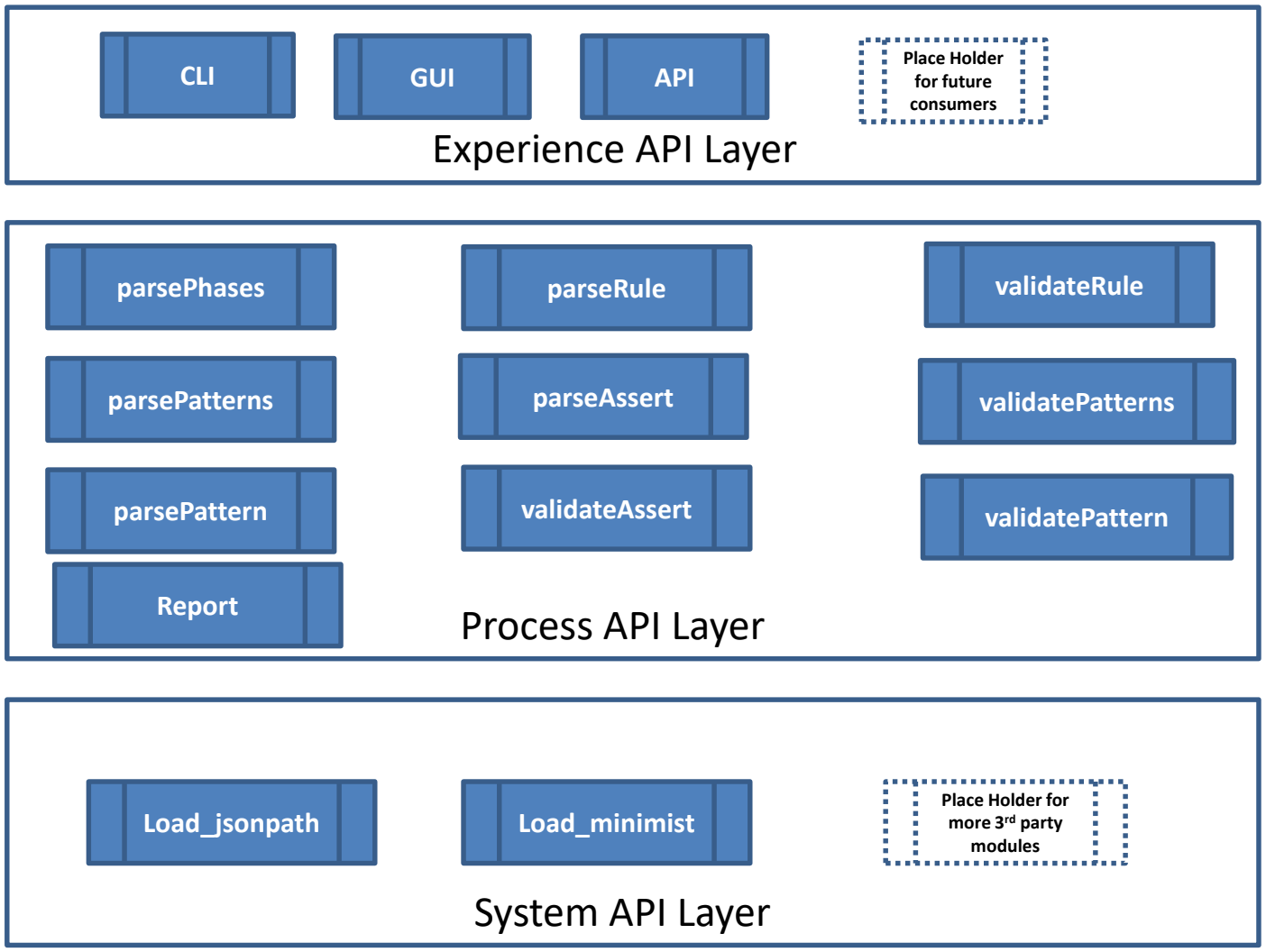


Figure 8

Report Highlights

```
var Report = function(){

    this.errors = [];
    this.warnings = [];
    this.validations =[];


}

Report.prototype.addError = function(instance, schema, attr,msg,detail){

    this.errors.push({
        schInstance : instance,
        schema : schema,
        attribute: attr,
        message : msg,
        detail : detail
    });

}

Report.prototype.addWarning = function(instance, schema, attr,msg,detail){

    this.warnings.push({
        schInstance : instance,
        schema : schema,
        attribute: attr,
        message : msg,
        detail : detail
    });

}

Report.prototype.addValidation = function(rule, context, assertionid, test, msg, result){

    this.validations.push({
        schRule : rule,
        ruleContext : context,
        assertionid: assertionid,
        assertionTest : test,
        message : msg,
        assertionValid : result
    });
```

Listing 10

Use Cases

- Command Line Interface - **CLI**
- Graphical User Interface – **GUI**
- Application Programming Interface – **API**
- **Frontend** and **Backend** Hybrid Validation
- **Syntax & Semantic** Validation
- Handling **Partial** Validation
- Handling **Variation** Document Versions
- Handling **Multiple Form Factors**
- **Assumptions & Limitations**
  - Assumes **implicit compliance** through implementation
  - No control over upstream systems
  - Some **dependency on host language**

Experimental Study

- **Data**
  - **Motivating example**
    - All examples described in motivating examples
  - **Store data example**
    - Popular data set to test JSON schema implementations
  - **IBM Schematron tutorial**
    - Popular tutorial to learn & test Schematron
    - Originall in XML
    - Translated all XML instance into JSON documents
    - Translated all Rules file into JSON rules files
    - Created it as a stand alone tutorial
- **Tests**
  - Jasmine
  - **~300**

Data Snippet


```
{
    "loan_data":{
        "loans":[
            {
                "loan_id":"1234567",
                "loan_type":"FHA",
                "customer_id":"JD689457",
                "data_time":"20100601120000",
                "amount":500000,
                "interest_rate":3.75,
                "prime_rate":3.25,
                "mip_rate":1.5,
                "down_payment":5,
                "loan_restricted":false,
                "escrow":true,
                "origination_id":"branch",
                "branch_id":"5463",
                "electronic":true,
                "email":"john.doe@gmail.com",
                "customer":{
                    "customer_id":"JD689457",
                    "customer_fname":"John",
                    "customer_lname":"Doe",
                    "customer_address":" 4 Way Loop, New York, NY 10038"

                        }
            },
            {
                "loan_id":"2111112",
                "loan_type":"FHA",
                "customer_id":"JD689457",
                "data_time":"20100601120000",
                "amount":500000,
                "interest_rate":3.75,
                "prime_rate":3.25,
                "mip_rate":1.8,
                "down_payment":5,
                "loan_restricted":false,
                "escrow":true,
                "origination_id":"branch",
                "branch_id":"5463",
                "electronic":true,
                "email":"john.doe@gmail.com",
                "customer":{
                    "customer_id":"JD689457",
                    "customer_fname":"John",
                    "customer_lname":"Doe",
                    "customer_address":" 4 Way Loop, New York, NY 10038"

                        }
            },
```


loandata_pattern_good2.json

Rules Snippet


```
    "phase":[
        {
            "id":"precheck",
            "active":["precheck_pattern"]
        },
        {
            "id":"newfha",
            "active":["newfha_pattern"]
        },



    {
      "id":"newfha_pattern",
      "title":"New FHA MIP Pattern",
      "abstract":false,
      "rule":[
        {
                "id":"rule-new",
              "abstract":false,
              "context": "$.loan_data.loans[?( @.loan_id > 2111111 && @.loan_type === 'FHA')]",
              "assert":[
                {
                  "id":"assertid31",
                  "test": "jp.query(contextNode, '$..mip_rate') >= 1.8 ",
                  "message": "Assert 31: New FHA loan can't have less than 1.8 percent mortgage insurance premium"
                }
              ]
        }
        ]
      },
```


loandata-rules_dissertation_pattern_good2.json

Command

```
C:\Users\DPS\Dropbox\workspaces\gitrepos\jsontron\bin>node JSONValidator -i ..\data\dissertation\pattern\loandata_pattern_good2.json -r ..\data\dissertation\pattern\loandata-rules_dissertation_pattern_good2.
json newfha -d
```

Output Report

```
Starting Semantic Validation .........
Parsing Pattern: newfha_pattern
1 Pattern(s) Requested. 1 Pattern(s) Processed. 0 Pattern(s)  Ignored.
**** THIS INSTANCE IS SEMANTICALLY VALID ****
Completed Semantic Validation .........
Total Errors Found: 0
Total Warnings Found: 0
Total Validations: 1
Total Failded Assertions: 0
```

Data Snippet

```
"loans":[
    {
        "loan_id":"1234567",
        "loan_type":"FHA",
        "customer_id":"JD689457",
        "data_time":"20100601120000",
        "amount":500000,
        "interest_rate":3.75,
        "prime_rate":3.25,
        "mip_rate":0,
        "down_payment":5,
        "loan_restricted":false,
        "escrow":true,
        "origination_id":"branch",
        "branch_id":"5463",
        "electronic":true,
        "email":"john.doe@gmail.com",
        "customer":{
            "customer_id":"JD689457",
            "customer_fname":"John",
            "customer_lname":"Doe",
            "customer_address":" 4 Way Loop, New York, NY 10038"

                    }


    },
```

loandata_dataForRules_bad1.json

Rules Snippet

```
31      "rule":[
32       {
33              "id":"rule-pre",
34            "abstract":false,
35            "context": "$.loan_data.loans[?(@.loan_type === 'FHA')]",
36            "assert":[
37             {
38               "id":"assertid31",
39               "test": "jp.query(contextNode, '$..mip_rate') > 0",
40               "message": "Assert 31: FHA loan can't have zero mortgage insurance premium"
41             }
42            ]
43       }
44       ]
```

loandata-rules_dissertation_rules_good1.json

Command

```
C:\Users\DPS\Dropbox\workspaces\gitrepos\jsontron\bin>node JSONValidator -i ..\data\dissertation\rules\loandata_dataForRules_bad1.json -r ..\data\dissertation\rules\loandata-rules_dissertation_rules_good1.json precheck -d
```

Output Report

```
Starting Semantic Validation .........
Parsing Pattern: precheck_pattern
1 Pattern(s) Requested. 1 Pattern(s) Processed. 0 Pattern(s)  Ignored.
**** THIS INSTANCE CONTAINS SEMANTIC VALIDATION ISSUES. PLEASE SEE FULL REPORT BY ENABLING DEBUG WITH -d OPTION ****
Completed Semantic Validation .........
Total Errors Found: 0
Total Warnings Found: 0
Total Validations: 1
Total Failded Assertions: 1
        assertionId: 'assertid31',
        assertionTest: 'jp.query(contextNode, \'$..mip_rate\') > 0',
        message: 'Assert 31: FHA loan can\'t have zero mortgage insurance premium',
        assertionValid: false } ],
```

Contributions

- Schematron based **framework** to specify semantic validation constraints
  - ‘schema’,’phase’, ‘pattern’, ‘rule’, and ‘assert’
- **Reusable Schema** for syntax validation of rules
- Reusable Semantic Validation **Rules Engine**
- Comprehensive **Reporting** Component
- Augmentation of syntax rules for
  - Progressive, partial, dynamic validation
- Schematron JSON Tutorials
- 300 Jasmine Unit Tests

Adaptation of Solution to Solve Similar Problems in Other Domains

- **API Gateway**
- **MDM** - Master Data Management
- **TDM** - Test Data Management
- **Big Data**
- **OVAL** for JSON
  - Open Vulnerability Assesment Language
- **Social Media** OVAL
- **NoSQL**, Document Oriented DBMS
- Enhancement for **action**

Potential Future Work

- Implement remaining Schematron **non core features**
- Switch **query language**
- Individual **APIs optimization**
- Experience APIS for main **platforms**
- **Streaming** JSON data processing
- **Action** instead of just message
- For Bigdata **SIMD** (Single Instruction, Multiple Data)
- **Serverless** Hosting of Validation Service
- **AI/Machine Learning** to to automatically generate and adjust rules

Conclusion

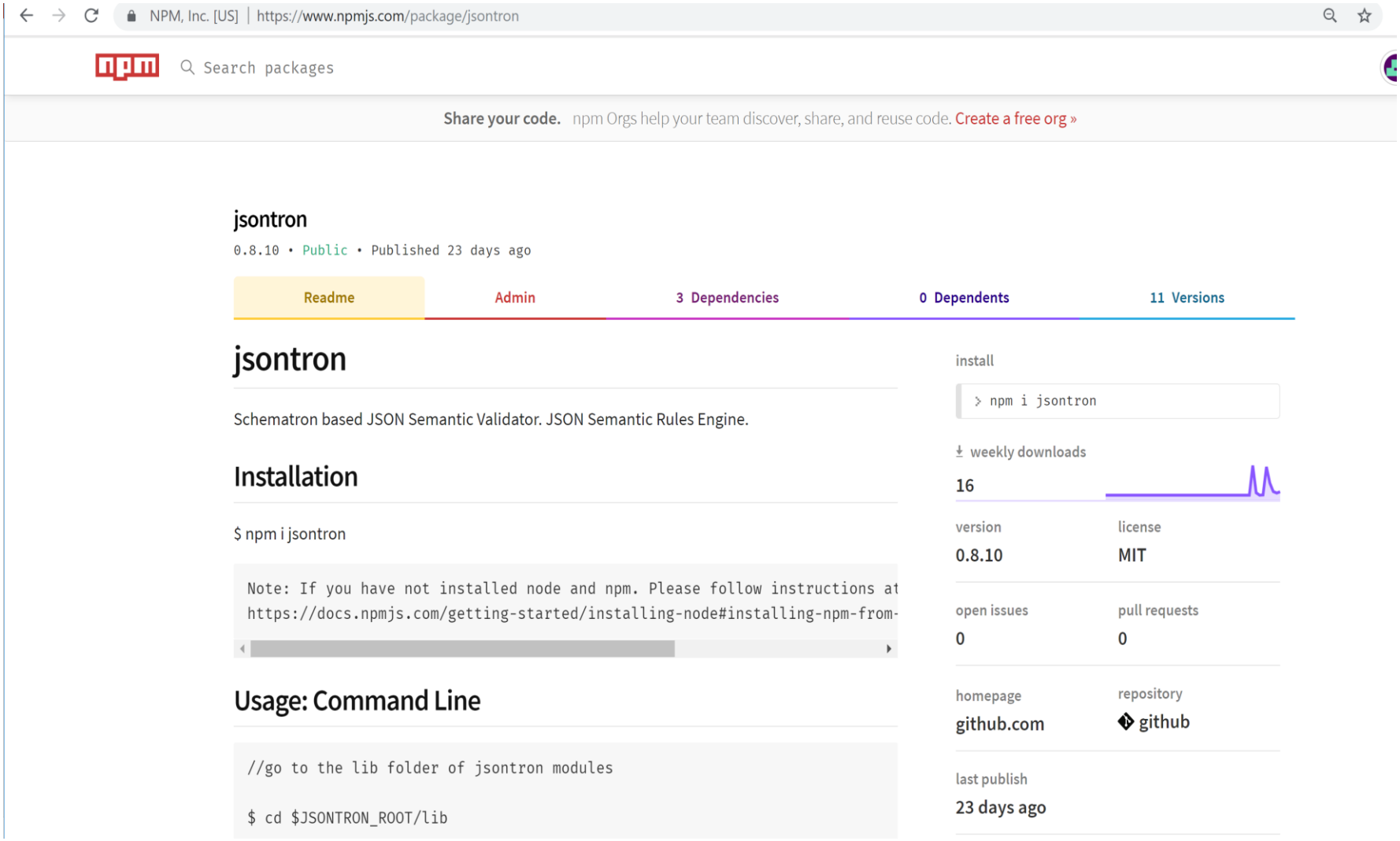
- JSON data format has serious **void in semantic** constraints specification and validation area
- In this study,
  - we created a Schematron based **framework** for constraints specification
  - A reusable JavaScript/Node **validator**
- We tested both of the components with almost 300 **tests**
- The component along with all its documentation and tests is hosted on **GitHub** and **NPM** registry
- Should serve as a ready to use system as well as **test bed** for further research in JSON semantic validation area

References

- [1] T. Redman, “Data: An unfolding quality disaster,” *Dm Rev.*, vol. 14, no. 8, pp. 21–23, 2004.
- [2] N. Chomsky, *Chomsky Hierarchy, Chomsky Normal Form*. General Books LLC, 2010.
- [3] M. W. Bovee, T. L. Roberts, and R. P. Srivastava, “Decisison Useful Financial Reporting Information Characteristics: An Empirical Validation of the Proposed FASB/IASB International Accounting Model,” *AMCIS 2009 Proc.*, p. 368, 2009.
- [4] L. P. English, *Improving Data Warehouse and Business Information Quality: Methods for Reducing Costs and Increasing Profits*. New York, New York, USA: John Wiley and Sons, Inc, 1999.
- [5] S. L. Meyers, “CIA Fires Officer Blamed in Bombing of Chinese Embassy,” *The New York Times*, p. A1, 09-Apr-2000.
- [6] M. S. Donaldson, J. M. Corrigan, L. T. Kohn, and others, *To err is human: building a safer health system*, vol. 6. National Academies Press, 2000.
- [7] P. Mcgeehan, “An Unlikely Clarion Calls for Change,” *The New York Times*, 16-Jun-2002.
- [8] M. R. Alvarez, S. Ansolabehere, E. Antonsson, and J. Bruck, “Voting, What Is, What Could Be,” *Rep. CALTECHMIT VOTING Technol. Proj.*, Jul. 2001.
- [9] S. Brunnermeier and S. A. Martin, *Interoperability cost analysis of the US automotive supply chain*. DIANE Publishing, 1999.
- [10]H. Tibbetts, “$3 Trillion Problem: Three Best Practices for Today’s Dirty Data Pandemic | Microservices Expo.” [Online]. Available: http://soa.sys-con.com/node/1975126. [Accessed: 02-Jul-2017].
- [11] F. Ted and M. Smith, “Measuring the Business Value of Data Quality,” Gartner, Analysis G00218962, Oct. 2011.
- [12] Experian Data Quality, “The Data Quality Benchmark Report,” Experian Information Solutions, Boston, MA, White Paper, Jan. 2015.
- [13] R. Singh, K. Singh, and others, “A descriptive classification of causes of data quality problems in data warehousing,” *Int. J. Comput. Sci. Issues*, vol. 7, no. 3, pp. 41–50, 2010.
- [14] V. K. Omachonu, J. E. Ross, and J. A. Swift, *Principles of total quality*. Boca Raton, Fla.: CRC Press, 2004.

Appendix

NPM


NPM, Inc. [US] | https://www.npmjs.com/package/jsontron
npm
Search packages
Share your code. npm Orgs help your team discover, share, and reuse code. Create a free org »
jsontron
0.8.10 • Public • Published 23 days ago
Readme
Admin
3 Dependencies
0 Dependents
11 Versions
jsontron
Schematron based JSON Semantic Validator. JSON Semantic Rules Engine.
Installation
$ npm i jsontron
Note: If you have not installed node and npm. Please follow instructions at
https://docs.npmjs.com/getting-started/installing-node#installing-npm-from-
Usage: Command Line
//go to the lib folder of jsontron modules
$ cd $JSONTRON_ROOT/lib
install
> npm i jsontron
weekly downloads
16
version
0.8.10
license
MIT
open issues
0
pull requests
0
homepage
github.com
repository
github
last publish
23 days ago

Schematron.com

schematron.com/2018/11/schematron-validation-of-json-data/

schematron

A language for making assertions about patterns found in XML documents

Schematron News Standards Hints Opinion

Schematron reimagined for JSON/JSONPath

Posted on November 7, 2018 by Rick Jelliffe

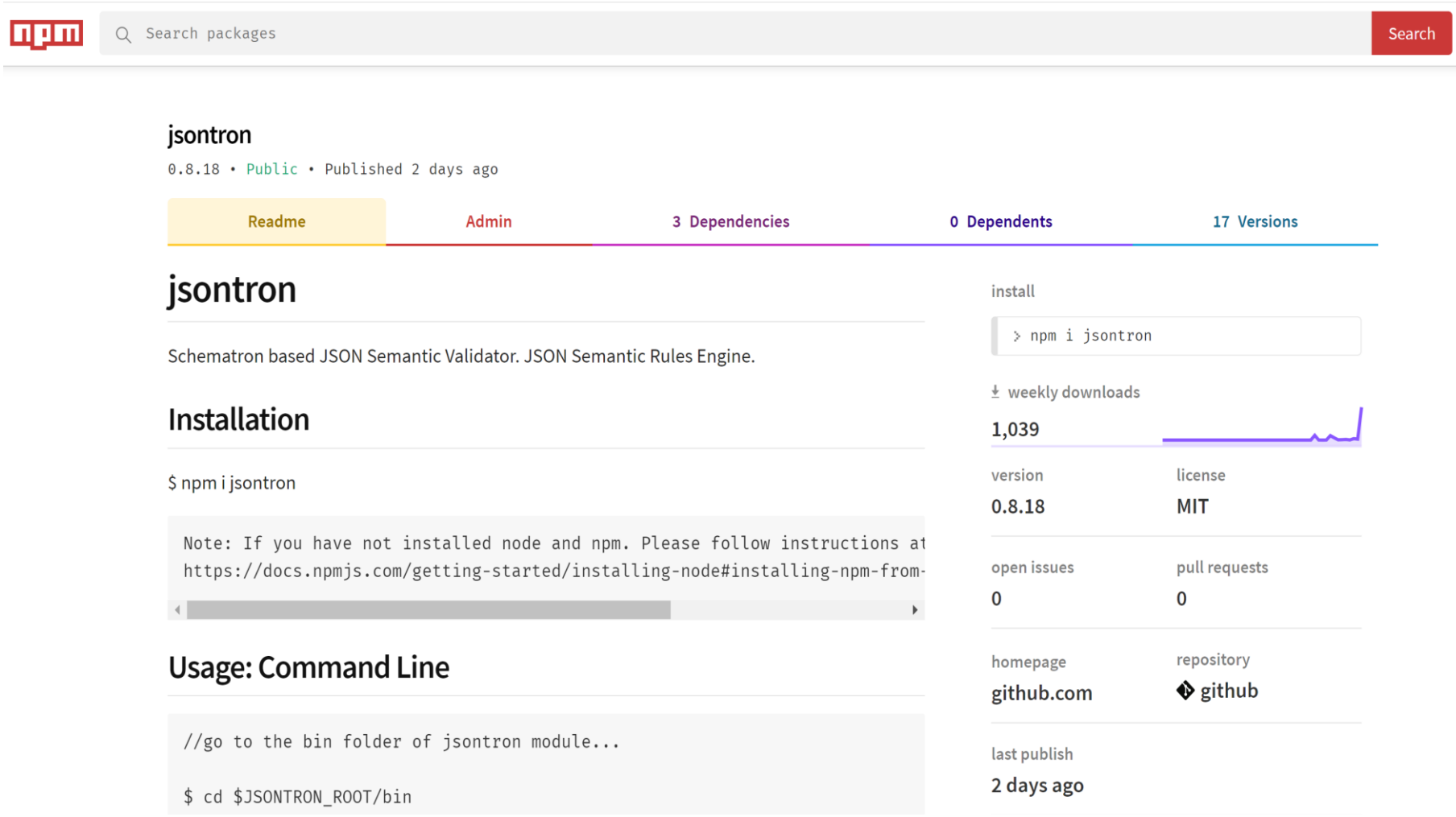
On GitHub you can find jsontron which is Schematron moved out of the XML/XSLT/XPath ecosystem and applied to the JSON/JavaScript/JSONPath ecosystem. What is particularly pleasing to me is that this seems to be a really full implementation of ISO Schematron, including phases (not abstract rules and abstract patterns, no biggie.)

It is written in JavaScript, takes a schema that is the JSON equivalent of a Schematron XML schema, and produces a JSON version of SVRL as output. It looks like something well worth the while for people who need it.

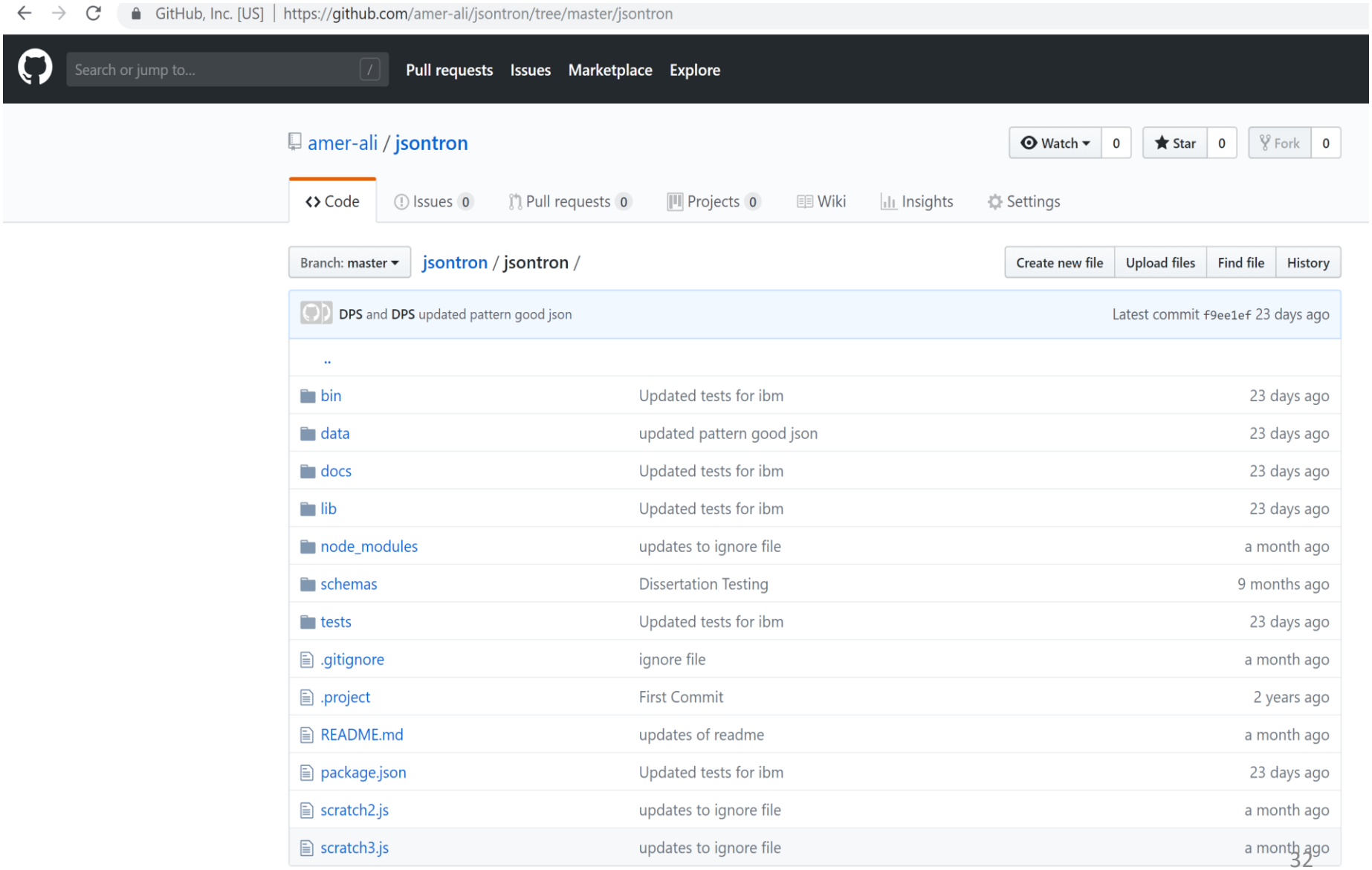
Amir Ali, who wrote it at Pace University as part of his studies, makes the point that JSON/JavaScript ecosystem systems need the OVAL (Open Vulnerability and Assement Language) validation regime as much as XML ecosystems do (perhaps more!). So a Schematron reimagined for JSON with no whiff of XML/XPath might be be sweeter for JSON/JavaScript developers.

Of course, not being XML, the schemas are not standard. But Amir Ali seems to have been very faithful to the structures and names of standard Schematron, so I guess it could be converted to

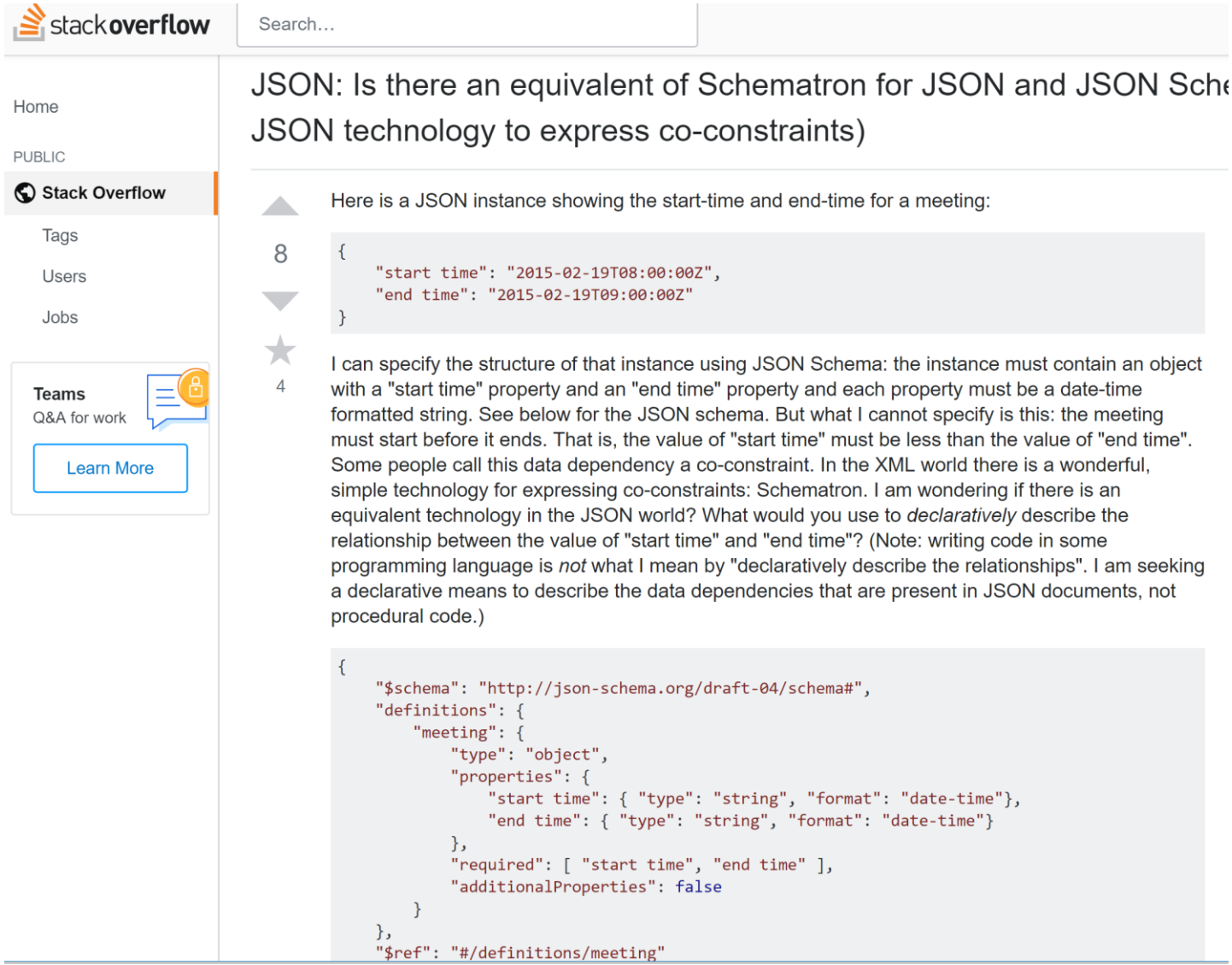
NPM


Search packages
Search
jsontron
0.8.18 • Public • Published 2 days ago
Readme
Admin
3 Dependencies
0 Dependents
17 Versions
jsontron
Schematron based JSON Semantic Validator. JSON Semantic Rules Engine.
Installation
$ npm i jsontron
Note: If you have not installed node and npm. Please follow instructions at
https://docs.npmjs.com/getting-started/installing-node#installing-npm-from-
Usage: Command Line
//go to the bin folder of jsontron module...
$ cd $JSONTRON_ROOT/bin
install
> npm i jsontron
weekly downloads
1,039
version
0.8.18
license
MIT
open issues
0
pull requests
0
homepage
github.com
repository
github
last publish
2 days ago

GitHub


GitHub, Inc. [US] | https://github.com/amer-ali/jsontron/tree/master/jsontron
Search or jump to...
Pull requests
Issues
Marketplace
Explore
amer-ali / jsontron
Watch 0
Star 0
Fork 0
Code
Issues 0
Pull requests 0
Projects 0
Wiki
Insights
Settings
Branch: master
jsontron / jsontron /
Create new file
Upload files
Find file
History
DPS and DPS updated pattern good json
Latest commit f9ee1ef 23 days ago
..
bin Updated tests for ibm 23 days ago
data updated pattern good json 23 days ago
docs Updated tests for ibm 23 days ago
lib Updated tests for ibm 23 days ago
node_modules updates to ignore file a month ago
schemas Dissertation Testing 9 months ago
tests Updated tests for ibm 23 days ago
.gitignore ignore file a month ago
.project First Commit 2 years ago
README.md updates of readme a month ago
package.json Updated tests for ibm 23 days ago
scratch2.js updates to ignore file a month ago
scratch3.js updates to ignore file a month ago

Stackoverflow



Stackoverflow

1

Sadly, the answer is no. JSON Schema allows you to validate the structure, and permitted values, but there are no mechanisms for validating sets of values, a'la Schematron.

The simplest way to solve this is to have another script in the pipeline which runs these kinds of

1

Yes.There is a **JSON Semantic Validator** based on **Schematron** available at:
https://www.npmjs.com/package/jsontron

It implements '*schema*', '*phase*', '*rule*', '*assert*' and reporting features of Schematron.

Here is when the original example of start time and end time was run through the validator:

good_time.json file contents:

```
{
"starttime": "2015-02-19T08:00:00Z",
"endtime": "2015-02-19T09:00:00Z"
}
```

bad_time.json file contents:

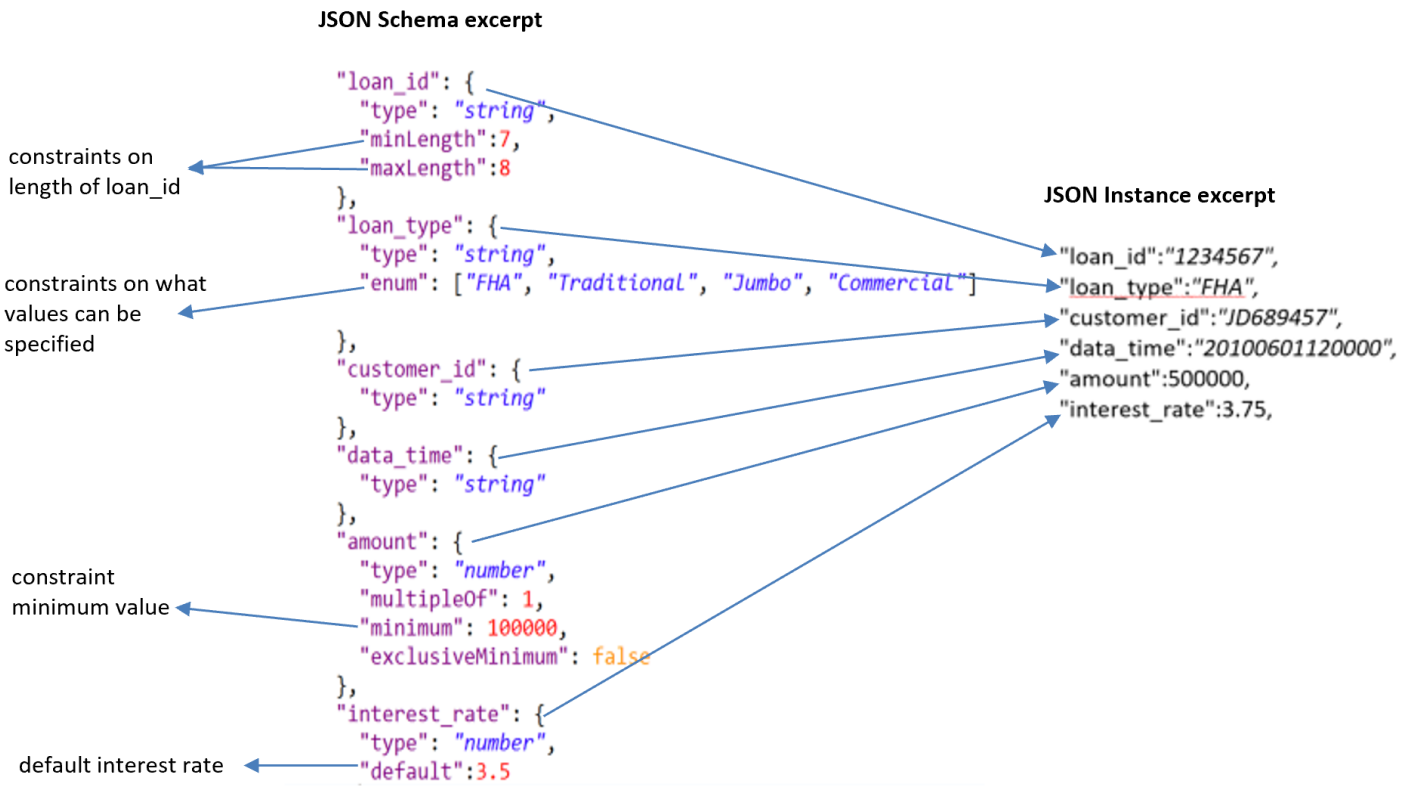
```
{
"starttime": "2015-02-19T09:00:00Z",
"endtime": "2015-02-19T08:00:00Z"
}
```

Schematron Rules file *meeting-times-rules.json* snippet:

```
        "rule":[
            {
            "context": "$",
            "assert":[
              {
                 "id":"start_stop_meeting_chec",
                 "test":"jp.query(contextNode, '$..starttime') < jp.query(contextNode, '$..er
                 "message": "Meeting cannot end before it starts"
              }
            ]
         }
        ]
```

JSON Schema

- JSON Schema is a **JSON-based** format for describing the structure of JSON data
- JSON Schema asserts what a JSON document must look like, ways to extract information from it, and how to interact with it
- It defines media type "**application/schema+json**"
- Unlike XML Schema, JSON Schema **is not an ISO standard** yet. It is an Internet Engineering Task **Force (IETF) draft**.
- The latest as of October, 2017 is draft 6 that was published on April 21st, 2017
- Since the latest draft is still being debated, this study will use **IETF draft version 4**



Listing 4

‘phase’ Element

JSON Schema Snippet

```
"phase": {
  "type": "array",
  "items": {
    "type": "object",
    "properties": {
      "id": {
        "type": "string"
      },
      "active": {
        "type": "array",
        "items": {
          "type": "string"
        }
      }
    },
    "required": [
      "id"
    ]
  }
```

Rules Snippet

```
"phase":[
    {
        "id":"phaseid1",
        "active":["patternid1"]
    },
      {
        "id":"phaseid2",
        "active":["patternid2"]
    }
],
```

‘pattern’ Element

JSON Schema Snippet

```
"pattern": {
  "type": "array",
  "items": {
    "type": "object",
    "properties": {
      "id": {
        "type": "string"
      },
      "title": {
        "type": "string"
      },
      "documents": {
        "type": "string"
      },
      "abstract": {
        "type": "boolean"
      },
```

Rules Snippet

```
"pattern":[
  {
    "id":"patternid1",
    "title":"pattern title",
    "documents":"pathValue",
    "abstract":false,
    "rule":[
      {
            "id":"rule1",
            "abstract":false,
            "context": "$.loan_data.loans.*",
            "assert":[
              {
                 "id":"assertid11",
```

‘rule’ Element

JSON Schema Definition

```
"rule": {
  "type": "array",
  "items": {
    "type": "object",
    "properties": {
      "id": {
        "type": "string"
      },
      "abstract": {
        "type": "boolean"
      },
      "context": {
        "type": "string"
      },
      "assert": {
```

Rules Snippet

```
"rule":[
  {
        "id":"rule1",
        "abstract":false,
        "context": "$.loan_data.loans.*",
        "assert":[
          {
```

The “context” expression in “**jsonpath**” states:
Select all loan objects from the loan_data json document.

Assertion Elements

JSON Schema Definition

```
"assert": {
  "type": "array",
  "items": {
    "type": "object",
    "properties": {
      "id": {
        "type": "string"
      },
      "test": {
        "type": "string"
      },
      "message": {
        "type": "string"
      }
    },
    "required": [
      "test",
      "message"
    ]
  }
},
```

Rules Snippet

```
"assert":[
 {
    "id":"assertid21",
    "test": "jp.query(contextNode,'$..amount') <= 500000",
    "message": "Assert 1: For FHA Loan, Amount cannot exceed $500K"
 },
```

<assert test="*test expression*"> Assertion message here </assert>

"test": <test goes here>
"message":< Assertion message here >

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "schema": {
      "type": "object",
      "properties": {
        "id": {
          "type": "string"
        },
        "title": {
          "type": "string"
        },
        "schemaVersion": {
          "type": "string"
        },
        "queryBinding": {
          "type": "string"
        },
        "defatulePhase": {
          "type": "string"
        },
        "phase": {
          "type": "array",
          "items": {
            "type": "object",
            "properties": {
              "id": {
                "type": "string"
              },
              "active": {
                "type": "array",
                "items": {
                  "type": "string"
                }
              }
            },
            "required": [
              "id"
            ]
          }
        },
        "pattern": {
          "type": "array",
          "items": {
            "type": "object",
            "properties": {
              "id": {
                "type": "string"
              },
              "title": {
                "type": "string"
              },
              "documents": {
                "type": "string"
              },
              "abstract": {
                "type": "boolean"
              },
              "rule": {
                "type": "array",
                "items": {
                  "type": "object",
                  "properties": {
                    "id": {
                      "type": "string"
                    },
                    "abstract": {
                      "type": "boolean"
                    },
                    "context": {
                      "type": "string"
                    },
                    "assert": {
                      "type": "array",
                      "items": {
                        "type": "object",
                        "properties": {
                          "id": {
                            "type": "string"
                          },
                          "test": {
                            "type": "string"
                          },
                          "message": {
                            "type": "string"
                          }
                        },
                        "required": [
                          "test",
                          "message"
                        ]
                      }
                    }
                  },
                  "required": [
                    "context",
                    "assert"
                  ]
                }
              },
              "required": [
                "id",
                "abstract"
              ]
            }
          }
        },
        "required": [
          "pattern"
        ]
      }
    },
    "required": [
      "schema"
    ]
  }
}
```

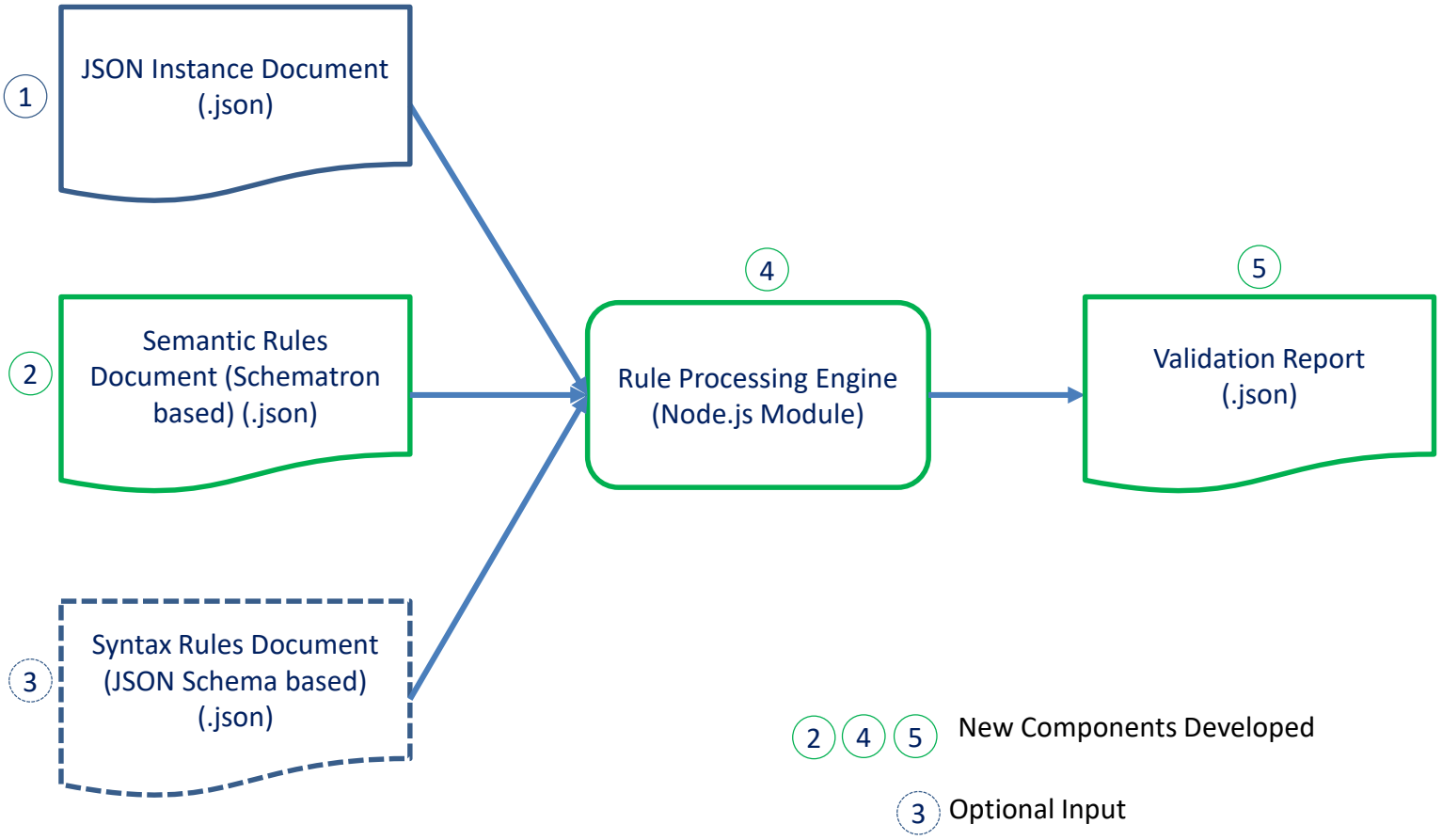
phase

```
$  node JSONValidator -i <json instance doc > -r <Schematron rule file>  phase1  phase2  phase3
```

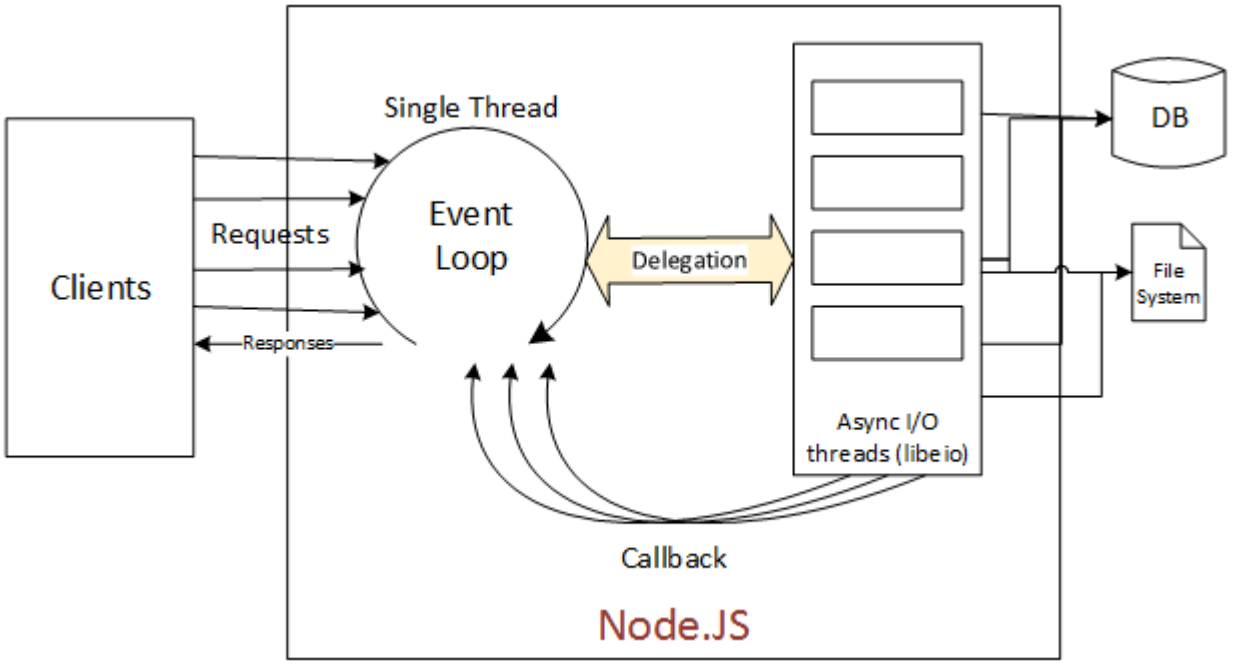
```
myReport = jsontron.JSONTRON.validate(schInstance, mySchRules, [‘phase1’, ‘phase2’, ‘phase3’])
```

```
"jp.query(contextNode,'$..amount') <= 500000"
```

IPO Pattern


1
JSON Instance Document (.json)
2
Semantic Rules Document (Schematron based) (.json)
3
Syntax Rules Document (JSON Schema based) (.json)
4
Rule Processing Engine (Node.js Module)
5
Validation Report (.json)
2 4 5 New Components Developed
3 Optional Input

Node.js Architecture



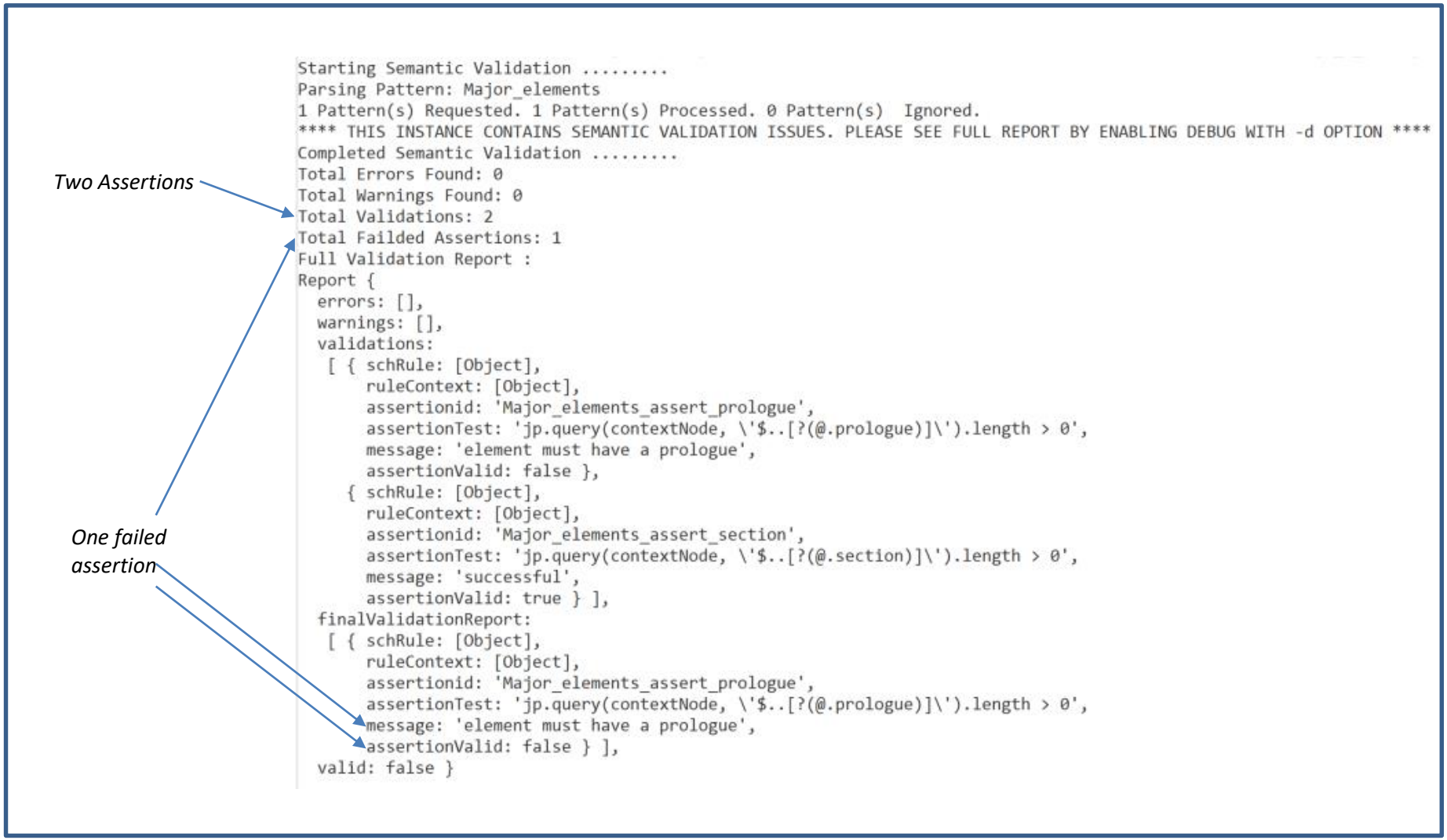
Courtesy: http://latestittrends.tumblr.com/

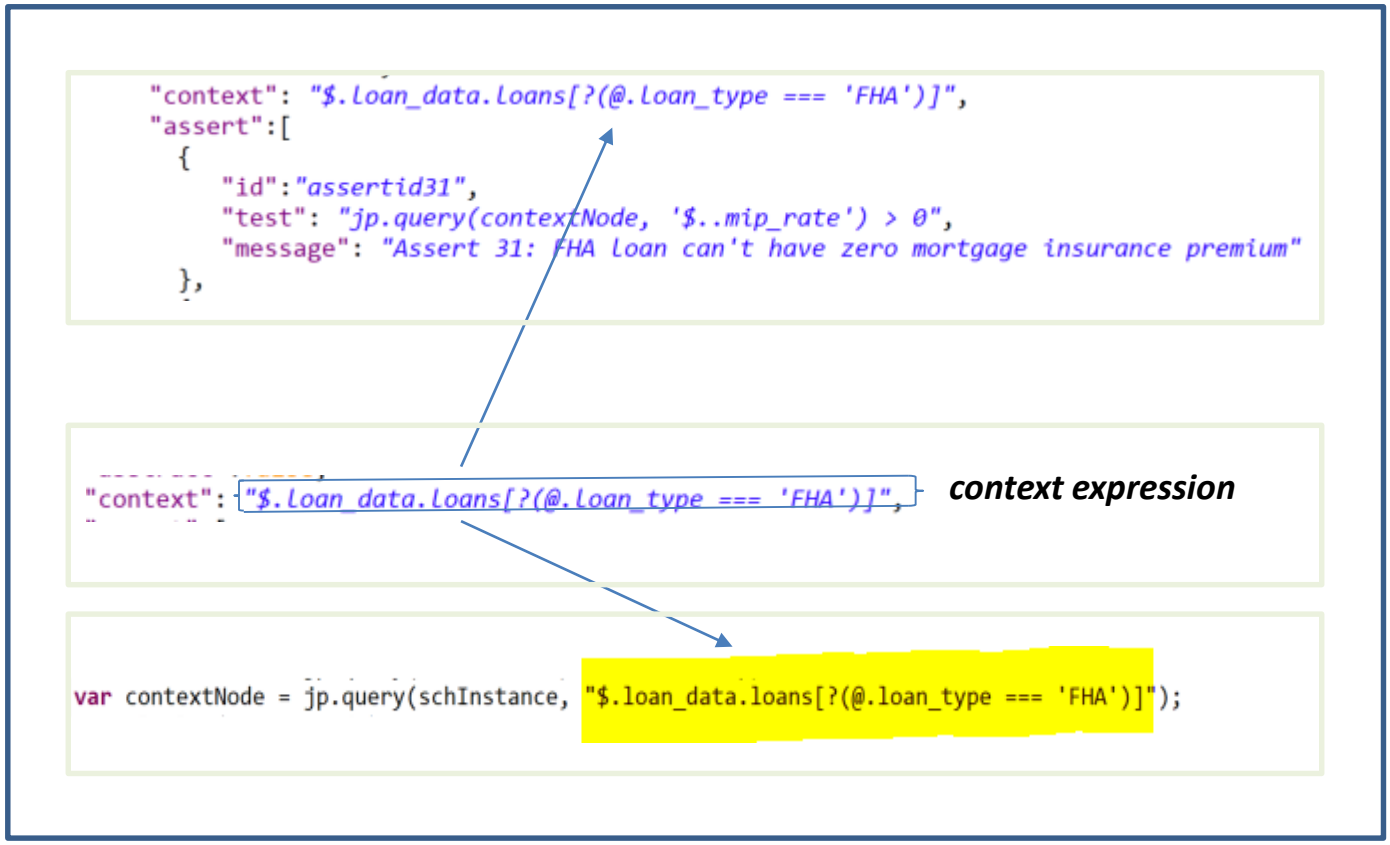
jsonpath

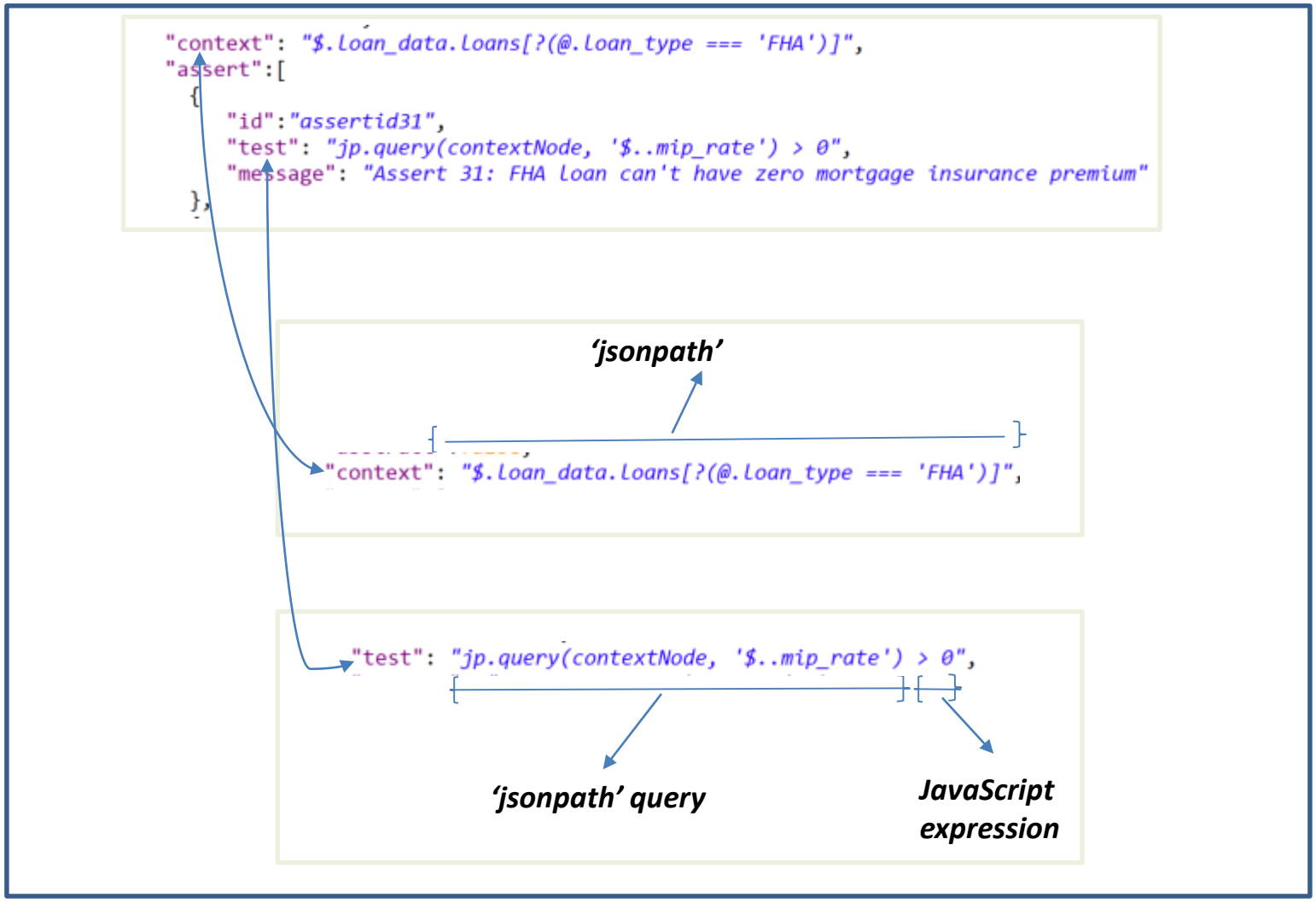
jp.query(obj, pathExpression[, count])

Find elements in `obj` matching `pathExpression`. Returns an array of elements that satisfy the provided JSONPath expression, or an empty array if none were matched. Returns only first `count` elements if specified.

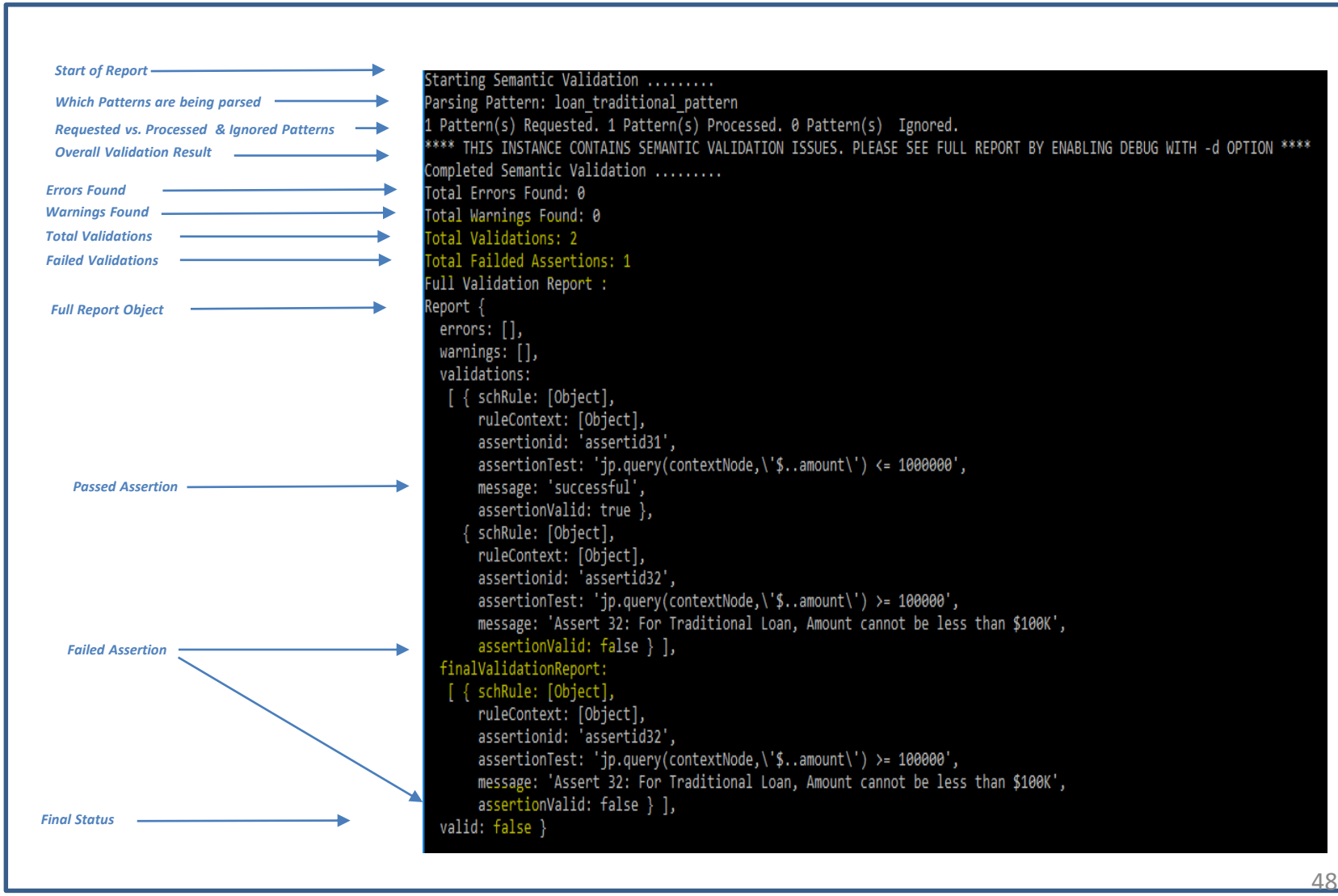
```
"jp.query(contextNode,'$..amount') <= 500000"
```


Two Assertions
One failed assertion
Starting Semantic Validation .....
Parsing Pattern: Major_elements
1 Pattern(s) Requested. 1 Pattern(s) Processed. 0 Pattern(s) Ignored.
**** THIS INSTANCE CONTAINS SEMANTIC VALIDATION ISSUES. PLEASE SEE FULL REPORT BY ENABLING DEBUG WITH -d OPTION ****
Completed Semantic Validation .....
Total Errors Found: 0
Total Warnings Found: 0
Total Validations: 2
Total Failded Assertions: 1
Full Validation Report :
Report {
errors: [],
warnings: [],
validations:
[{ schRule: [Object],
ruleContext: [Object],
assertionid: 'Major_elements_assert_prologue',
assertionTest: 'jp.query(contextNode, \'$..[?(@.prologue)]\').length > 0',
message: 'element must have a prologue',
assertionValid: false },
{ schRule: [Object],
ruleContext: [Object],
assertionid: 'Major_elements_assert_section',
assertionTest: 'jp.query(contextNode, \'$..[?(@.section)]\').length > 0',
message: 'successful',
assertionValid: true }],
finalValidationReport:
[{ schRule: [Object],
ruleContext: [Object],
assertionid: 'Major_elements_assert_prologue',
assertionTest: 'jp.query(contextNode, \'$..[?(@.prologue)]\').length > 0',
message: 'element must have a prologue',
assertionValid: false }],
valid: false }


"context": "$.loan_data.loans[?(@.loan_type === 'FHA')]",
"assert":[
{
"id":"assertid31",
"test": "jp.query(contextNode, '$..mip_rate') > 0",
"message": "Assert 31: FHA loan can't have zero mortgage insurance premium"
},
"context": "$.loan_data.loans[?(@.loan_type === 'FHA')]",
context expression
var contextNode = jp.query(schInstance, "$.loan_data.loans[?(@.loan_type === 'FHA')]");



"context": "$.loan_data.loans[?(@.loan_type === 'FHA')]",
"assert":[
{
"id":"assertid31",
"test": "jp.query(contextNode, '$..mip_rate') > 0",
"message": "Assert 31: FHA loan can't have zero mortgage insurance premium"
},


Start of Report
Which Patterns are being parsed
Requested vs. Processed & Ignored Patterns
Overall Validation Result
Errors Found
Warnings Found
Total Validations
Failed Validations
Full Report Object
Passed Assertion
Failed Assertion
Final Status
Starting Semantic Validation .....
Parsing Pattern: loan_traditional_pattern
1 Pattern(s) Requested. 1 Pattern(s) Processed. 0 Pattern(s) Ignored.
**** THIS INSTANCE CONTAINS SEMANTIC VALIDATION ISSUES. PLEASE SEE FULL REPORT BY ENABLING DEBUG WITH -d OPTION ****
Completed Semantic Validation .....
Total Errors Found: 0
Total Warnings Found: 0
Total Validations: 2
Total Failded Assertions: 1
Full Validation Report :
Report {
errors: [],
warnings: [],
validations:
[{ schRule: [Object],
ruleContext: [Object],
assertionid: 'assertid31',
assertionTest: 'jp.query(contextNode,\'$..amount\') <= 1000000',
message: 'successful',
assertionValid: true },
{ schRule: [Object],
ruleContext: [Object],
assertionid: 'assertid32',
assertionTest: 'jp.query(contextNode,\'$..amount\') >= 100000',
message: 'Assert 32: For Traditional Loan, Amount cannot be less than $100K',
assertionValid: false }],
finalValidationReport:
[{ schRule: [Object],
ruleContext: [Object],
assertionid: 'assertid32',
assertionTest: 'jp.query(contextNode,\'$..amount\') >= 100000',
message: 'Assert 32: For Traditional Loan, Amount cannot be less than $100K',
assertionValid: false }],
valid: false }