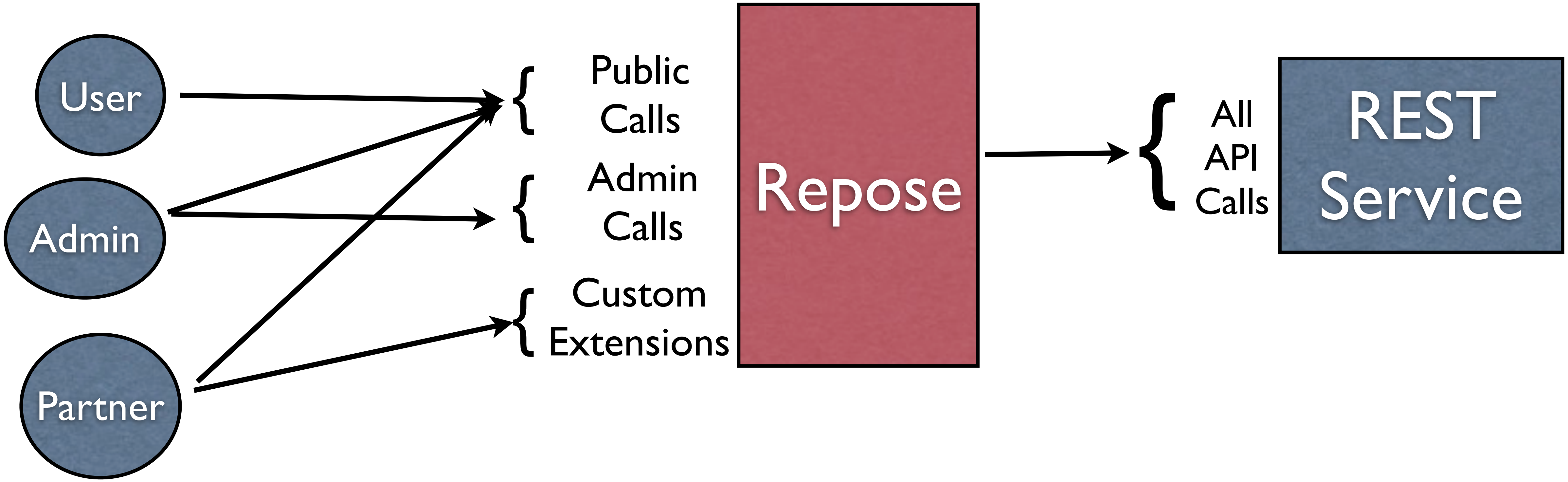
Contract Scope Filter

Two Use Cases

- Keep unauthorized API calls from going through - Contract Scope
- API coverage

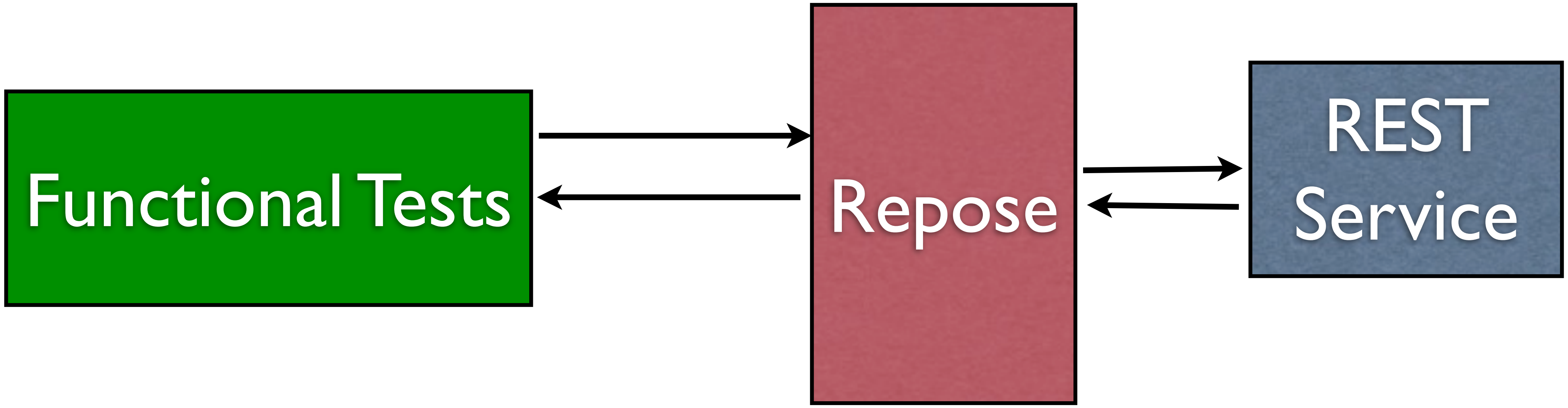
Contract Scope



Choose a different set of calls based on Group or Role

High Level Authorization

API Coverage

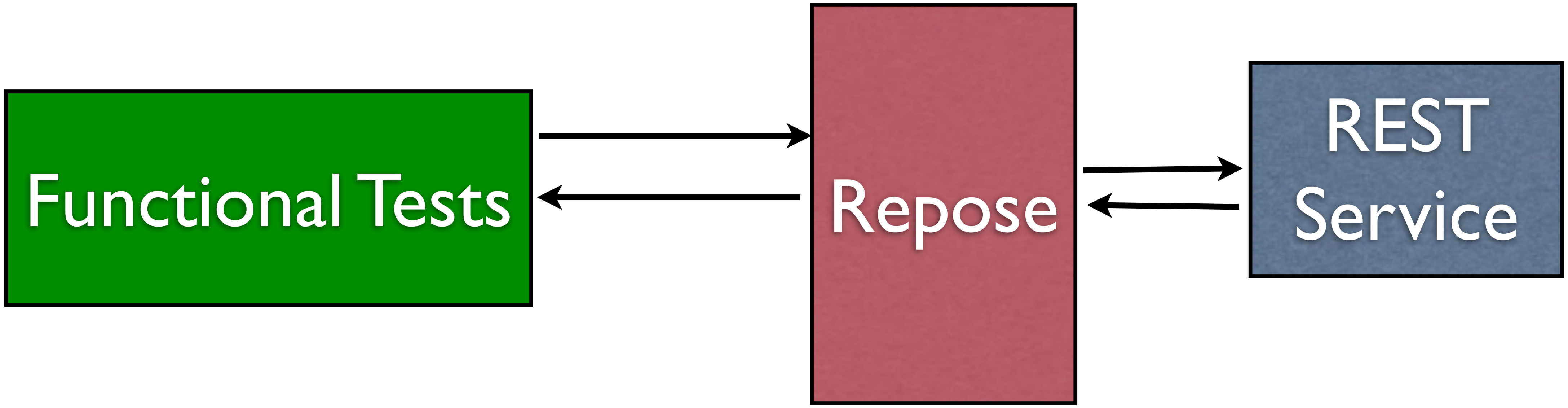


Functional Test of REST Service...

But how do you know the tests are correct?

How do you know, you’ve exercised the entire contract?

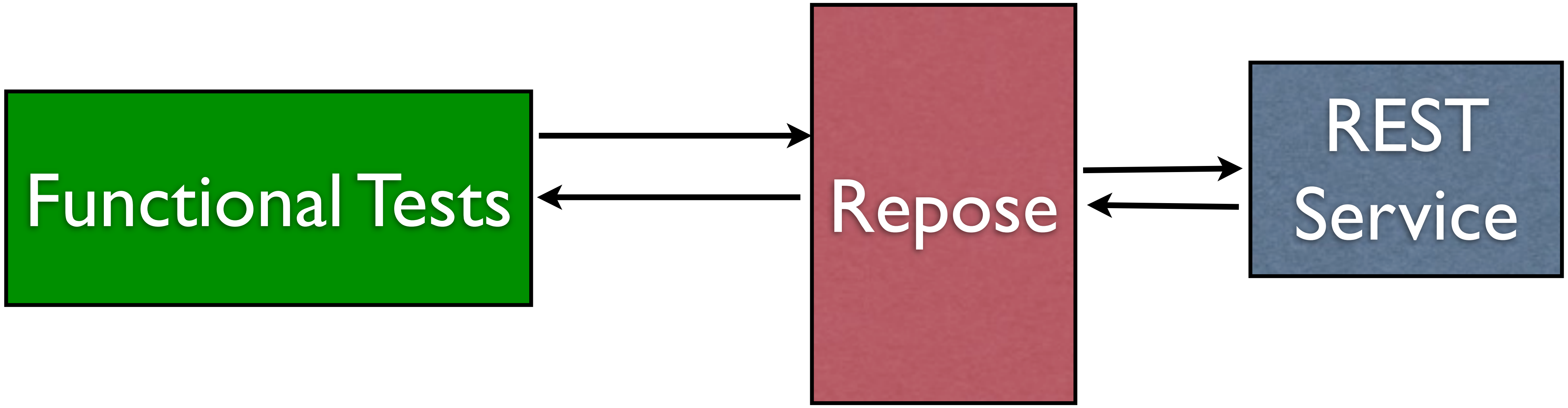
API Coverage



Repose Checks all Inputs/Output for conformance

Detail Checks involving Headers, Params, Methods, URI, etc.

API Coverage



Gives an exact coverage info to client
[Your tests exercise 89% of all methods...]

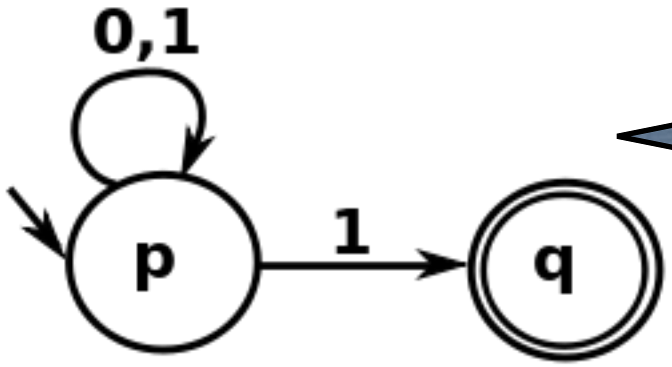
...and measures conformance to the contract...

Both Cases are very similar

- Need a machine processable description of what the API does.
- If we can process a WADL we can solve for both cases

Main Idea: Build a NFA

- NFA : A Nondeterministic finite automata
- A state machine used to identify a "language"



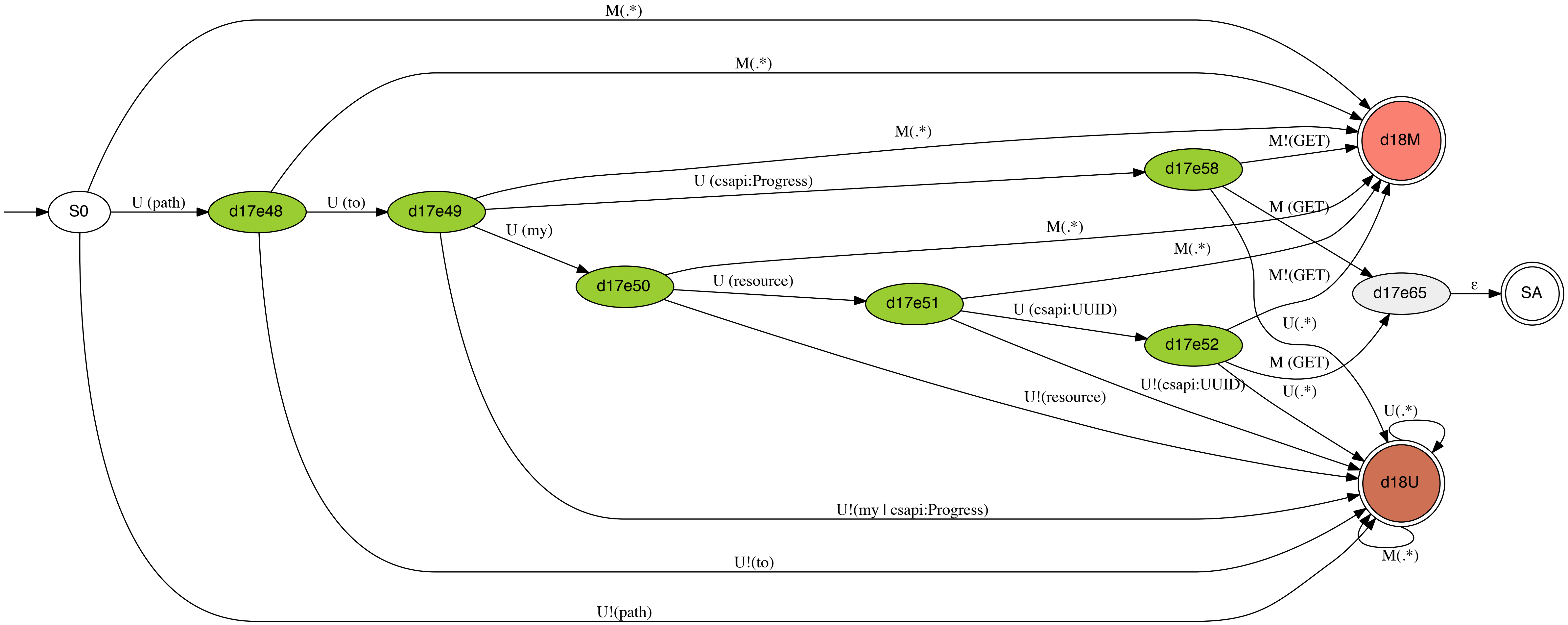
Accepts all binary input that ends in 1

NFAs

- Equivalent to REGExes -- these compile to NFAs -- or DFAs (deterministic cousins)
- Idea: The subset of HTTP that follows our contract is a language that can be accepted by an NFA.
- Not a traditional NFA -- different inputs URI, Method, Parameter etc.
- Transitions based on Regex, Schema Acceptance, etc .. an NFA of NFAs

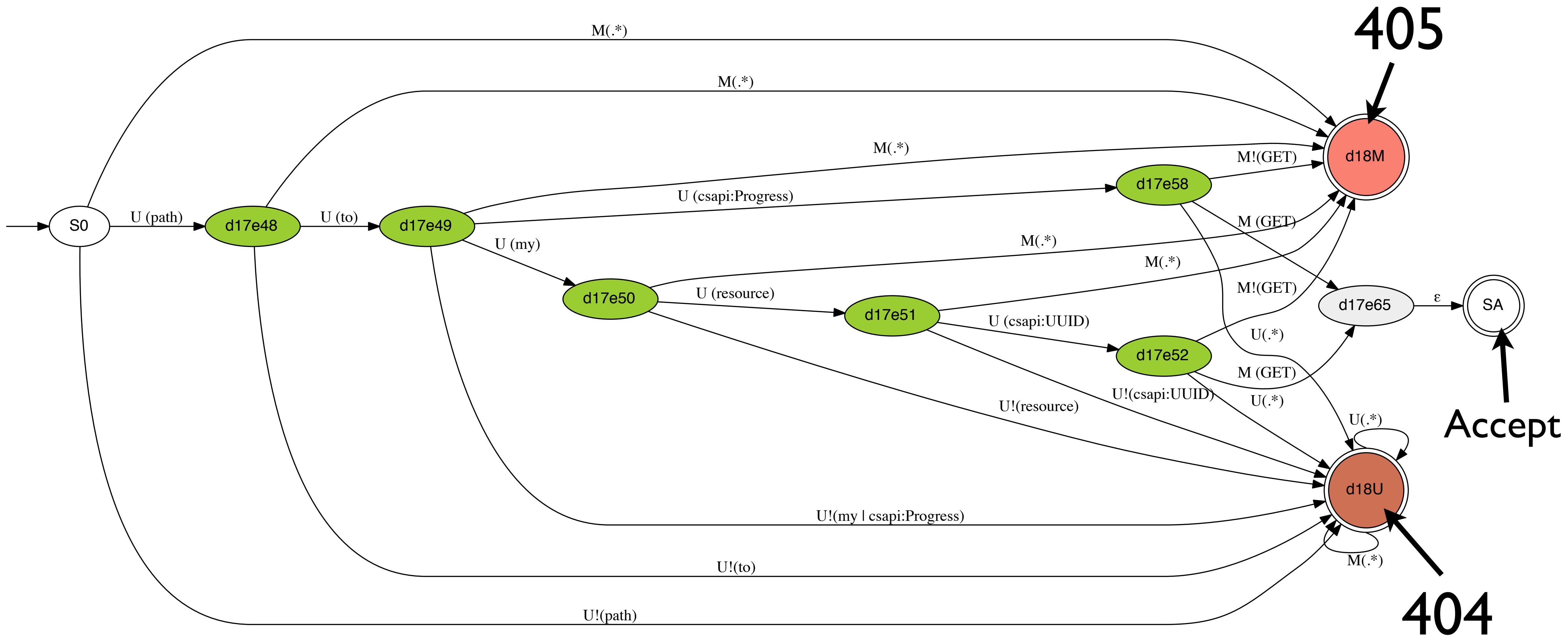
Given Any WADL...

- We can build a state machine that can accept the contract:



Given Any WADL...

- Errors are accept states, corresponding to errors codes 404, 405...



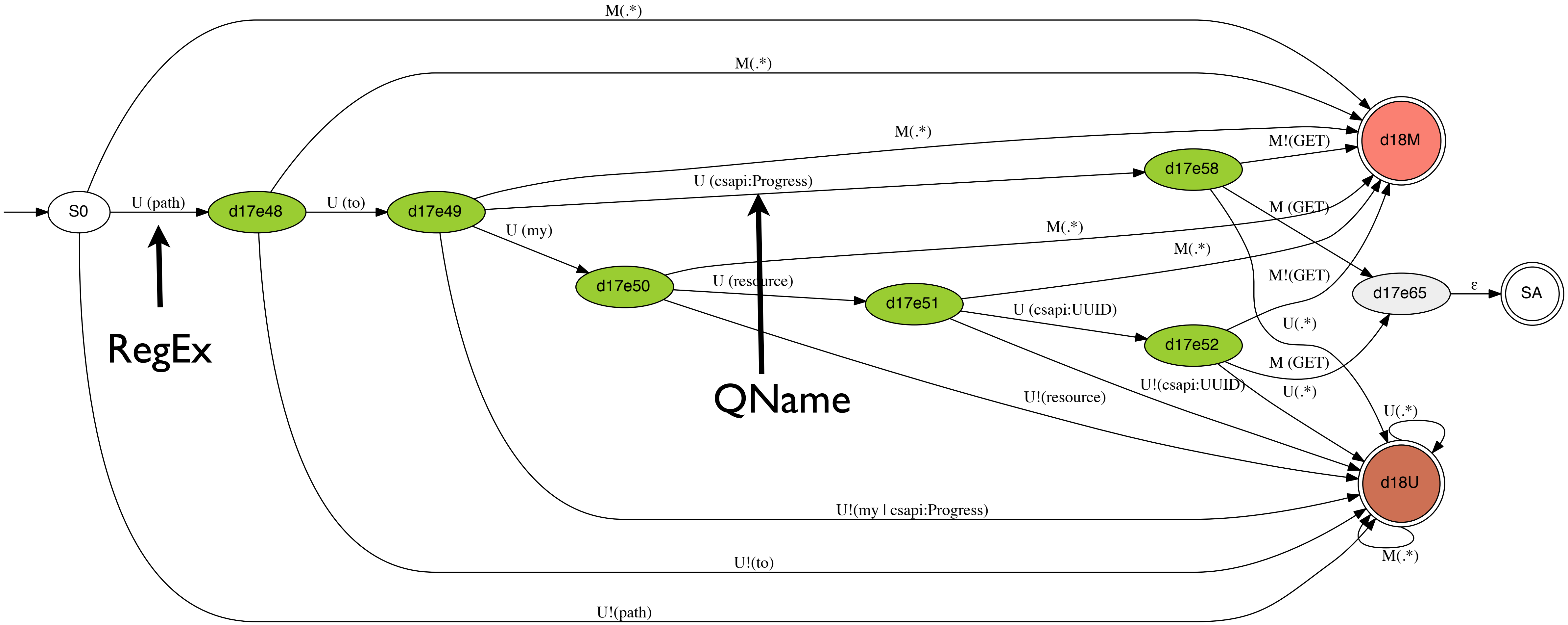
405

Accept

404

Given Any WADL...

- Transition based on RegEx, Simple type QName, etc.

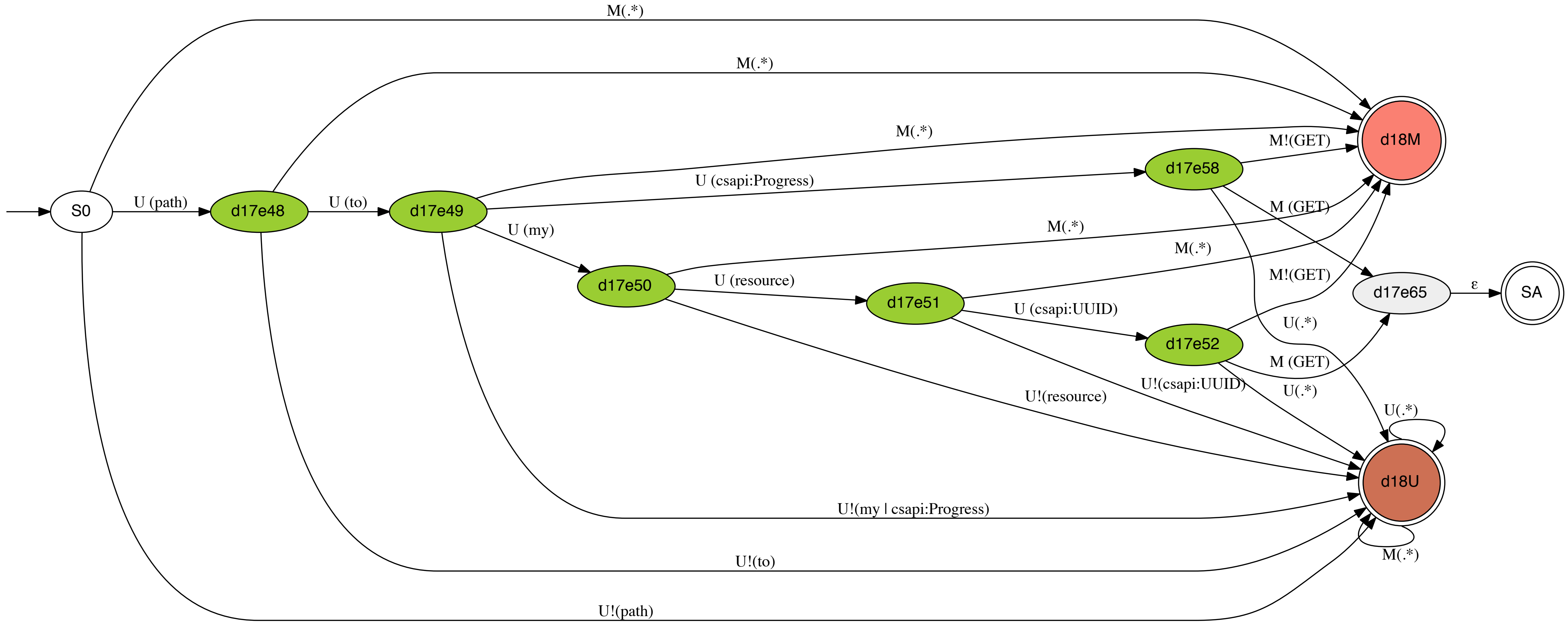


RegEx

QName

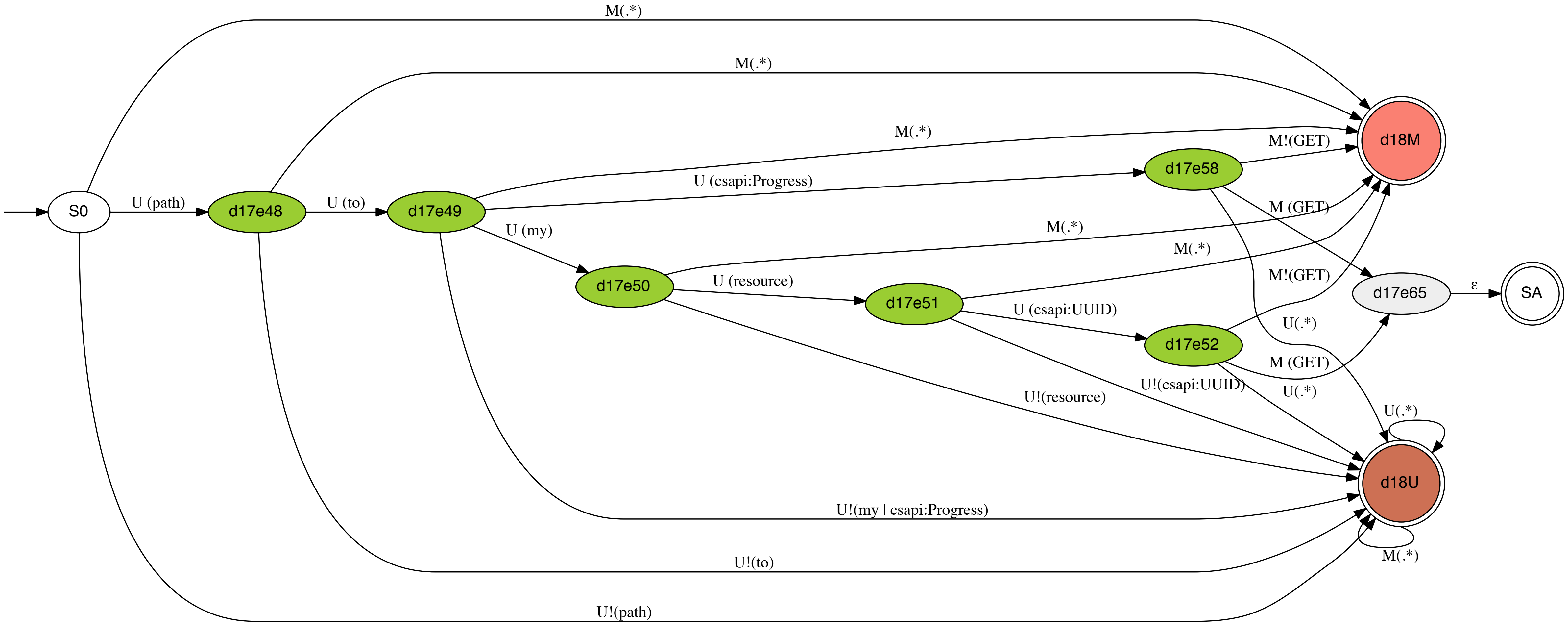
Given Any WADL...

- We can tell exactly, at what point an Error Occurred...



Given Any WADL...

- State Machine is immutable -- thus thread safe...
- Reporting done through async dispatches after the machine checks a request.



Example

```
<application xmlns="http://wadl.dev.java.net/2009/02"
  xmlns:csapi="http://docs.openstack.org/compute/api/v1.1">
  <grammars>
    <schema elementFormDefault="qualified"
      attributeFormDefault="unqualified"
      xmlns="http://www.w3.org/2001/XMLSchema"
      xmlns:csapi="http://docs.openstack.org/compute/api/v1.1"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://docs.openstack.org/compute/api/v1.1">
      <simpleType name="Progress">
        <restriction base="xsd:int">
          <minInclusive value="0"/>
          <maxInclusive value="100" />
        </restriction>
      </simpleType>
      <simpleType name="UUID">
        <restriction base="xsd:string">
          <length value="36" fixed="true"/>
          <pattern value="[a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}"/>
        </restriction>
      </simpleType>
    </schema>
  </grammars>
  <resources base="https://test.api.openstack.com">
    <resource id="uuid" path="path/to/my/resource/{uuid}">
      <param name="uuid" style="template" type="csapi:UUID"/>
      <method href="#getMethod" />
    </resource>
    <resource id="progress" path="path/to/{progress}">
      <param name="progress" style="template" type="csapi:Progress"/>
      <method href="#getMethod" />
    </resource>
  </resources>
  <method id="getMethod" name="GET">
    <response status="200 203"/>
  </method>
</application>
```

Example

```
<application xmlns="http://wadl.dev.java.net/2009/02"
  xmlns:csapi="http://docs.openstack.org/compute/api/v1.1">
  <grammars>
    <schema elementFormDefault="qualified"
      attributeFormDefault="unqualified"
      xmlns="http://www.w3.org/2001/XMLSchema"
      xmlns:csapi="http://docs.openstack.org/compute/api/v1.1"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://docs.openstack.org/compute/api/v1.1">
      <simpleType name="Progress">
        <restriction base="xsd:int">
          <minInclusive value="0"/>
          <maxInclusive value="100" />
        </restriction>
      </simpleType>
      <simpleType name="UUID">
        <restriction base="xsd:string">
          <length value="36" fixed="true"/>
          <pattern value="[a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}"/>
        </restriction>
      </simpleType>
    </schema>
  </grammars>
  <resources base="https://test.api.openstack.com">
    <resource id="uuid" path="path/to/my/resource/{uuid}">
      <param name="uuid" style="template" type="csapi:UUID"/>
      <method href="#getMethod" />
    </resource>
    <resource id="progress" path="path/to/{progress}">
      <param name="progress" style="template" type="csapi:Progress"/>
      <method href="#getMethod" />
    </resource>
  </resources>
  <method id="getMethod" name="GET">
    <response status="200 203"/>
  </method>
</application>
```

} Progress is an integer from 0 to 100

Example

```
<application xmlns="http://wadl.dev.java.net/2009/02"
  xmlns:csapi="http://docs.openstack.org/compute/api/v1.1">
  <grammars>
    <schema elementFormDefault="qualified"
      attributeFormDefault="unqualified"
      xmlns="http://www.w3.org/2001/XMLSchema"
      xmlns:csapi="http://docs.openstack.org/compute/api/v1.1"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://docs.openstack.org/compute/api/v1.1">
      <simpleType name="Progress">
        <restriction base="xsd:int">
          <minInclusive value="0"/>
          <maxInclusive value="100" />
        </restriction>
      </simpleType>
      <simpleType name="UUID">
        <restriction base="xsd:string">
          <length value="36" fixed="true"/>
          <pattern value="[a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}"/>
        </restriction>
      </simpleType>
    </schema>
  </grammars>
  <resources base="https://test.api.openstack.com">
    <resource id="uuid" path="path/to/my/resource/{uuid}">
      <param name="uuid" style="template" type="csapi:UUID"/>
      <method href="#getMethod" />
    </resource>
    <resource id="progress" path="path/to/{progress}">
      <param name="progress" style="template" type="csapi:Progress"/>
      <method href="#getMethod" />
    </resource>
  </resources>
  <method id="getMethod" name="GET">
    <response status="200 203"/>
  </method>
</application>
```

} UUID is a string with 36 chars that

is accepted by the given RegEX

Example

```xml
<application xmlns="http://wadl.dev.java.net/2009/02"
   xmlns:csapi="http://docs.openstack.org/compute/api/v1.1">
   <grammars>
      <schema elementFormDefault="qualified"
         attributeFormDefault="unqualified"
         xmlns="http://www.w3.org/2001/XMLSchema"
         xmlns:csapi="http://docs.openstack.org/compute/api/v1.1"
         xmlns:xsd="http://www.w3.org/2001/XMLSchema"
         targetNamespace="http://docs.openstack.org/compute/api/v1.1">
         <simpleType name="Progress">
            <restriction base="xsd:int">
               <minInclusive value="0"/>
               <maxInclusive value="100" />
            </restriction>
         </simpleType>
         <simpleType name="UUID">
            <restriction base="xsd:string">
               <length value="36" fixed="true"/>
               <pattern value="[a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}"/>
            </restriction>
         </simpleType>
      </schema>
   </grammars>
   <resources base="https://test.api.openstack.com">
      <resource id="uuid" path="path/to/my/resource/{uuid}">
         <param name="uuid" style="template" type="csapi:UUID"/>
         <method href="#getMethod" />
      </resource>
      <resource id="progress" path="path/to/{progress}">
         <param name="progress" style="template" type="csapi:Progress"/>
         <method href="#getMethod" />
      </resource>
   </resources>
   <method id="getMethod" name="GET">
      <response status="200 203"/>
   </method>
</application>
```

} GET allowed on
/path/to/my/resource/{UUID}

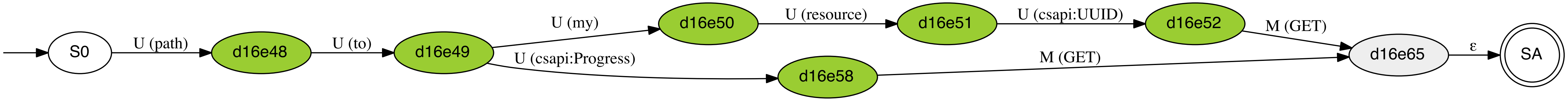
Example

```xml
<application xmlns="http://wadl.dev.java.net/2009/02"
  xmlns:csapi="http://docs.openstack.org/compute/api/v1.1">
  <grammars>
    <schema elementFormDefault="qualified"
      attributeFormDefault="unqualified"
      xmlns="http://www.w3.org/2001/XMLSchema"
      xmlns:csapi="http://docs.openstack.org/compute/api/v1.1"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://docs.openstack.org/compute/api/v1.1">
      <simpleType name="Progress">
        <restriction base="xsd:int">
          <minInclusive value="0"/>
          <maxInclusive value="100" />
        </restriction>
      </simpleType>
      <simpleType name="UUID">
        <restriction base="xsd:string">
          <length value="36" fixed="true"/>
          <pattern value="[a-f0-9]{8}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{4}-[a-f0-9]{12}"/>
        </restriction>
      </simpleType>
    </schema>
  </grammars>
  <resources base="https://test.api.openstack.com">
    <resource id="uuid" path="path/to/my/resource/{uuid}">
      <param name="uuid" style="template" type="csapi:UUID"/>
      <method href="#getMethod" />
    </resource>
    <resource id="progress" path="path/to/{progress}">
      <param name="progress" style="template" type="csapi:Progress"/>
      <method href="#getMethod" />
    </resource>
  </resources>
  <method id="getMethod" name="GET">
    <response status="200 203"/>
  </method>
</application>
```

} GET allowed on
/path/to/{progress}

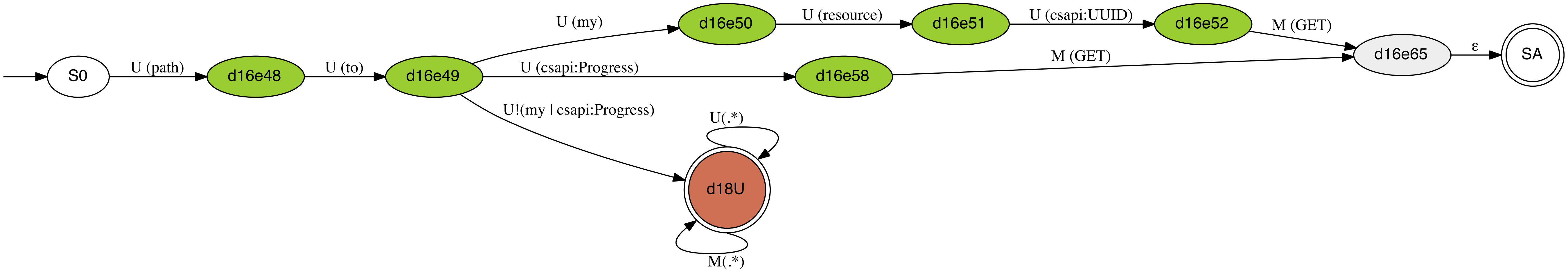
Example

- The WADL is converted to a state machine like this (Error Stages removed for clarity)



Example

- The filter can reject requests that don't match with meaningful Error Messages.
- For example a GET on /path/to/hello, it can say 404, /path/to/{hello}, hello is not an integer and is not "my"



Challenge: WADLs are hard for machines to parse, human friendly, you can express the same WADL in many different ways:

```
<application xmlns="http://wadl.dev.java.net/2009/02">
   <grammars/>
   <resources base="https://test.api.openstack.com">
      <resource path="path/to/my/resource/">
         <method name="GET">
            <response status="200 203"/>
         </method>
         <method name="DELETE">
            <response status="200"/>
         </method>
      </resource>
   </resources>
</application>
```

Challenge: WADLs are hard for machines to parse, human friendly, you can express the same WADL in many different ways:

```xml
<application xmlns="http://wadl.dev.java.net/2009/02">
    <grammars/>
    <resources base="https://test.api.openstack.com">
        <resource path="path">
            <resource path="to">
                <resource path="my">
                    <resource path="resource">
                        <method name="GET">
                            <response status="200 203"/>
                        </method>
                        <method name="DELETE">
                            <response status="200"/>
                        </method>
                    </resource>
                </resource>
            </resource>
        </resource>
    </resources>
</application>
```

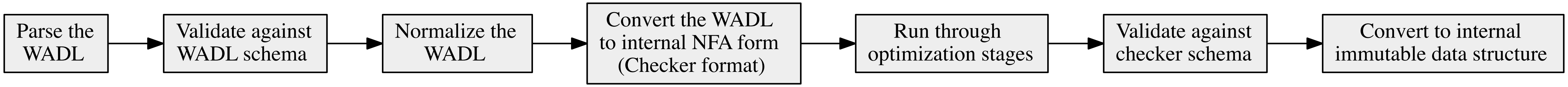
Challenge: WADLs are hard for machines to parse, human friendly, you can express the same WADL in many different ways:

```
<application xmlns="http://wadl.dev.java.net/2009/02">
   <grammars/>
   <resources base="https://test.api.openstack.com">
      <resource path="path/to/my">
         <resource path="resource">
            <method name="GET">
               <response status="200 203"/>
            </method>
            <method name="DELETE">
               <response status="200"/>
            </method>
         </resource>
      </resource>
   </resources>
</application>
```

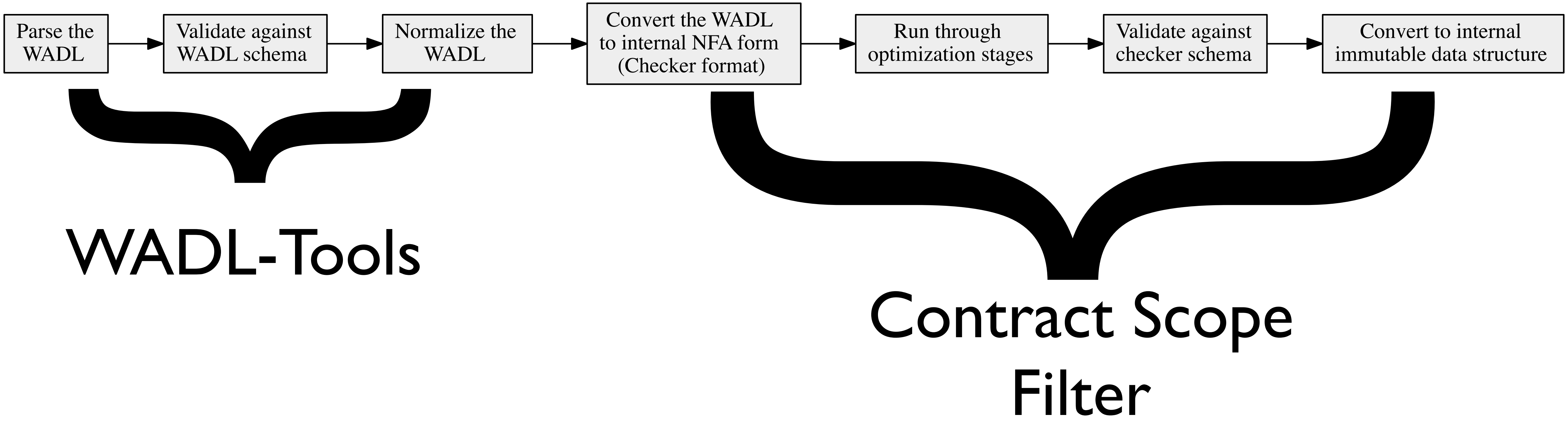
WADL-Tools to the Rescue

- The WADL can be normalized by WADL-Tools so that it is easily parseable by the Contract Scope Filter
- WADL-Tools, in prod, used for our API documentation

WADLs are processed in a pipeline...


Parse the WADL
Validate against WADL schema
Normalize the WADL
Convert the WADL to internal NFA form (Checker format)
Run through optimization stages
Validate against checker schema
Convert to internal immutable data structure

WADLs are processed in a pipeline...

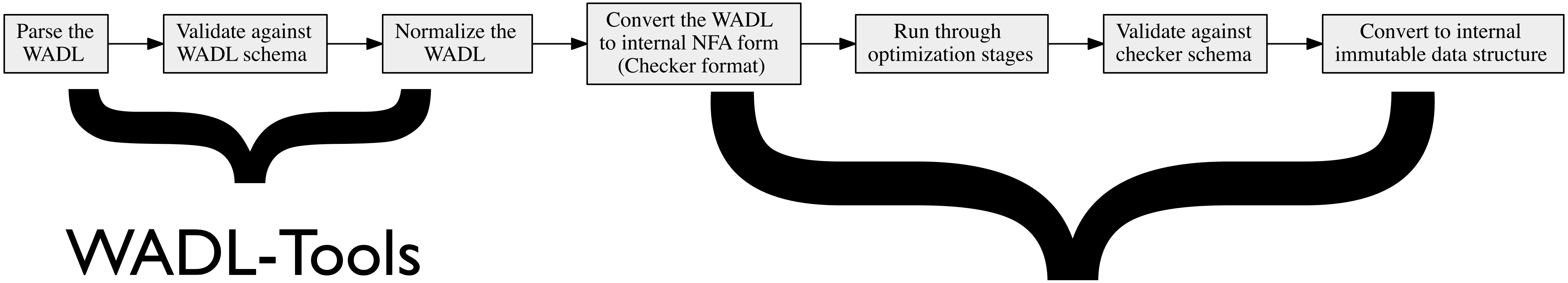


Parse the WADL → Validate against WADL schema → Normalize the WADL → Convert the WADL to internal NFA form (Checker format) → Run through optimization stages → Validate against checker schema → Convert to internal immutable data structure

WADL-Tools

Contract Scope Filter

WADLs are processed in a pipeline...



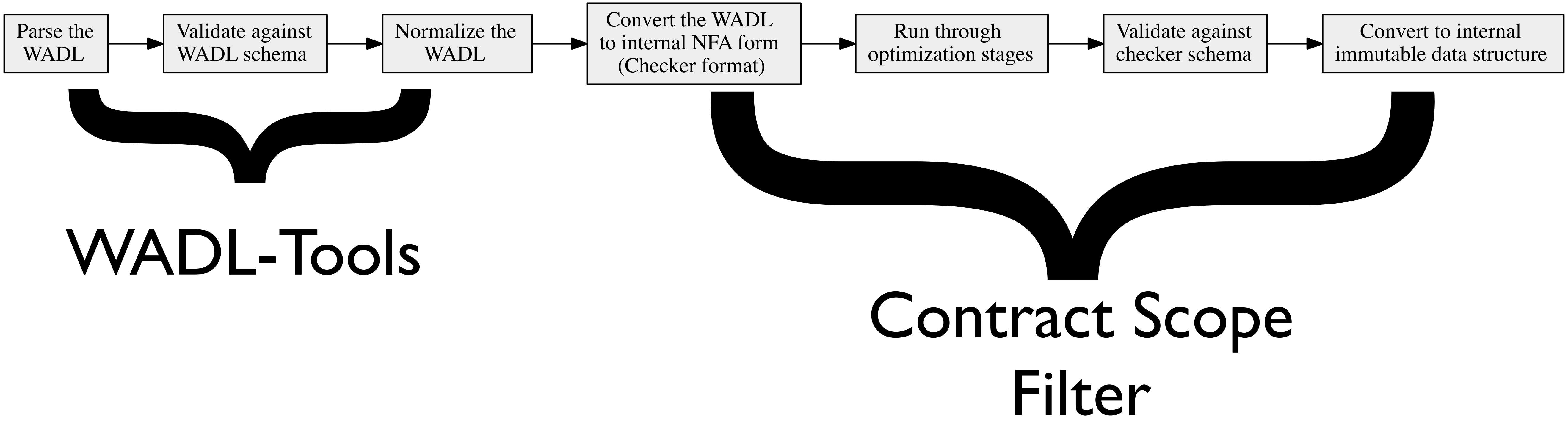
- Parse the WADL
- Validate against WADL schema
- Normalize the WADL
- Convert the WADL to internal NFA form (Checker format)
- Run through optimization stages
- Validate against checker schema
- Convert to internal immutable data structure

WADL-Tools

Contract Scope Filter

This is a preprocessing step...
...performed when the filter is started.

WADLs are processed in a pipeline...



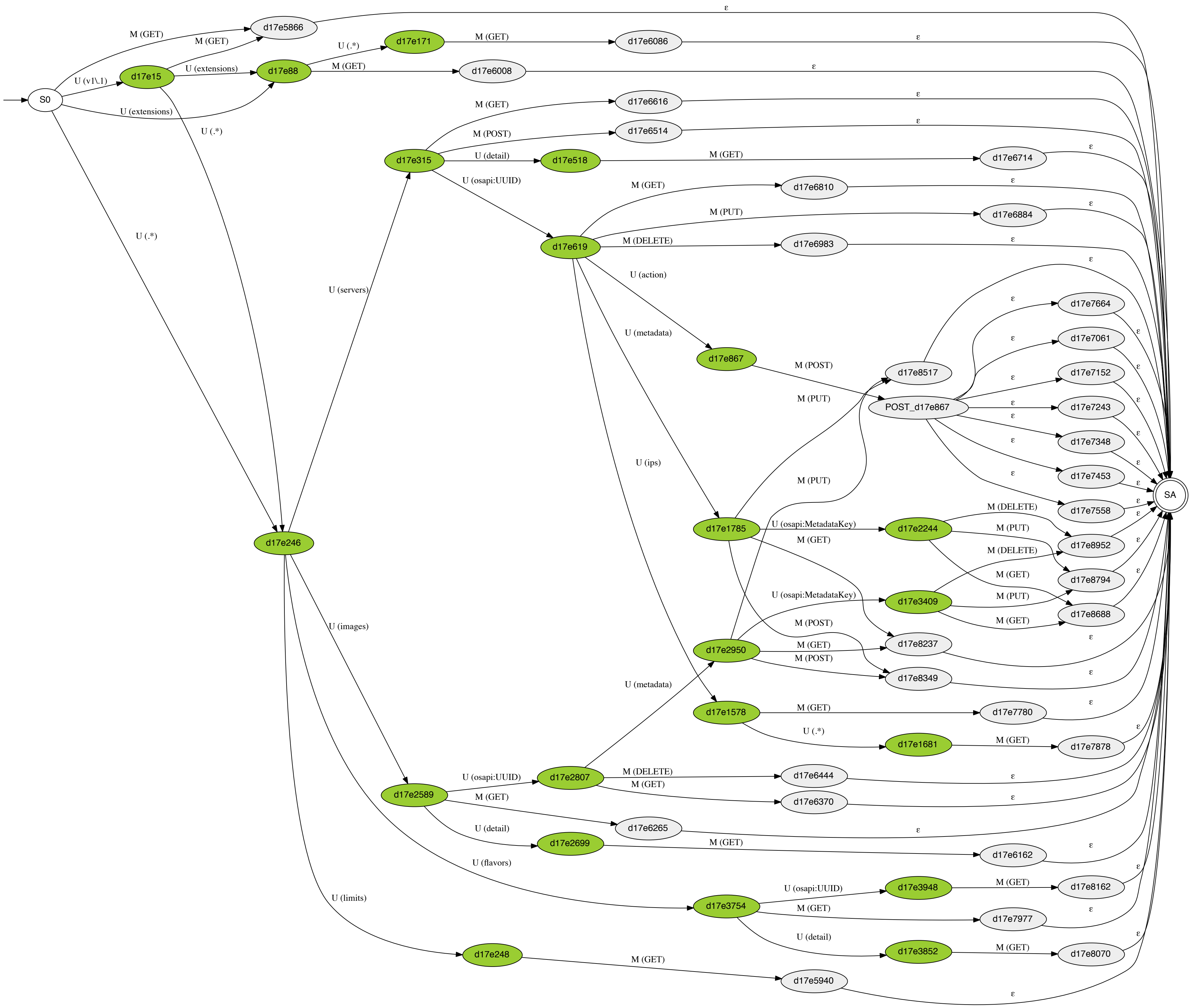
Parse the WADL → Validate against WADL schema → Normalize the WADL → Convert the WADL to internal NFA form (Checker format) → Run through optimization stages → Validate against checker schema → Convert to internal immutable data structure

WADL-Tools

Contract Scope Filter

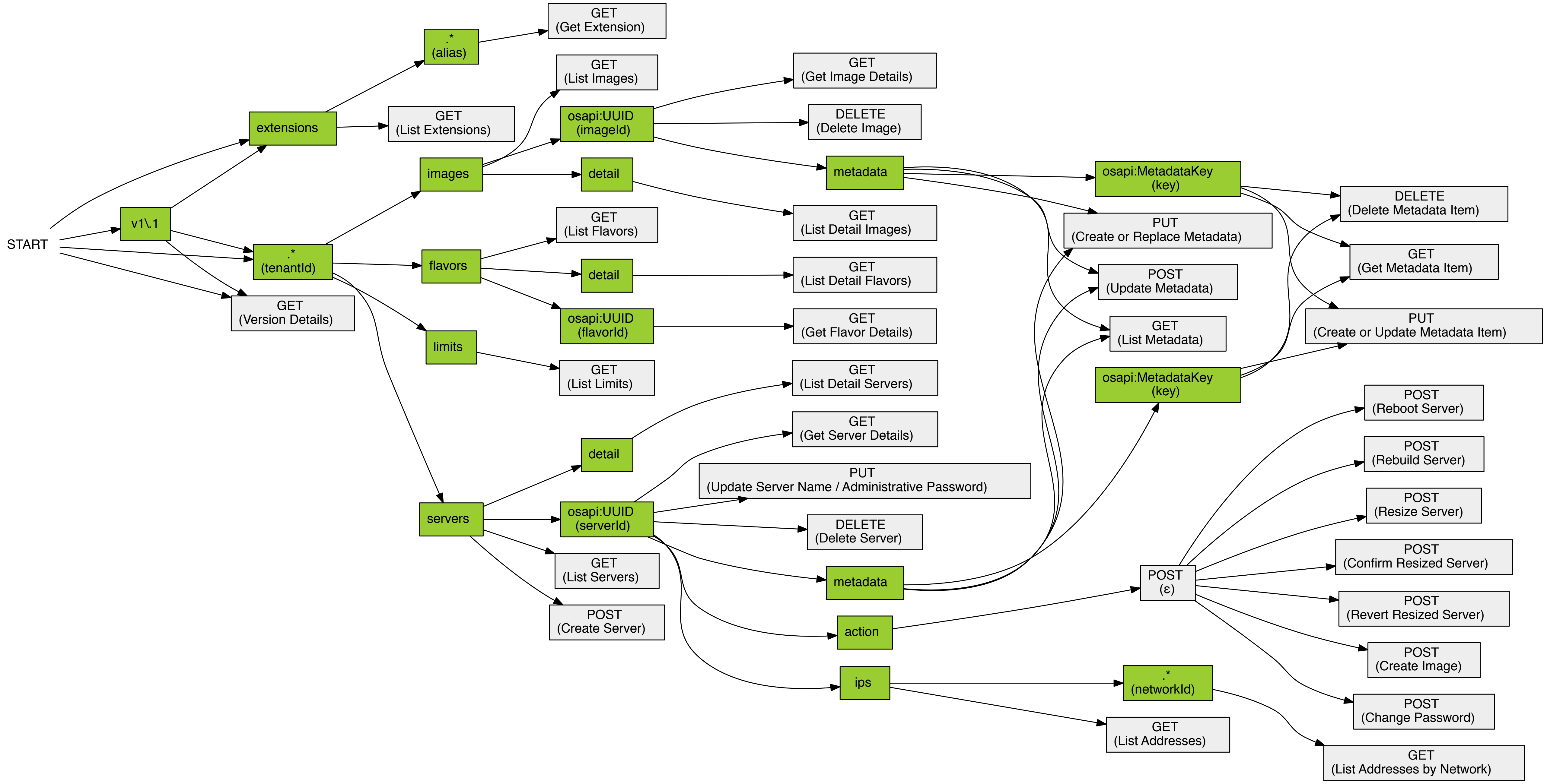
Once the immutable data structure is created it is used to validate each request.

Compute API


ε
M (GET)
d17e5866
M (GET)
U (.*)
d17e171
M (GET)
d17e6086
ε
U (v1\.1)
d17e15
U (extensions)
d17e88
M (GET)
d17e6008
ε
S0
U (extensions)
M (GET)
d17e6616
ε
U (.*)
M (POST)
d17e6514
ε
d17e315
U (detail)
d17e518
M (GET)
d17e6714
ε
U (osapi:UUID)
M (GET)
d17e6810
ε
U (.*)
M (PUT)
d17e6884
ε
d17e619
M (DELETE)
d17e6983
ε
U (action)
ε
U (servers)
ε
d17e7664
U (metadata)
ε
d17e7061
ε
d17e867
M (POST)
d17e8517
ε
d17e7152
ε
M (PUT)
POST_d17e867
ε
d17e7243
ε
ε
U (ips)
d17e7348
ε
ε
d17e7453
ε
M (PUT)
SA
ε
d17e7558
M (DELETE)
d17e1785
U (osapi:MetadataKey)
d17e2244
M (PUT)
d17e8952
d17e246
M (GET)
M (DELETE)
ε
M (GET)
d17e8794
U (osapi:MetadataKey)
d17e3409
M (PUT)
ε
U (images)
M (GET)
d17e8688
M (POST)
ε
d17e2950
M (GET)
d17e8237
M (POST)
d17e8349
U (metadata)
ε
d17e1578
M (GET)
d17e7780
ε
U (.*)
d17e1681
M (GET)
d17e7878
ε
U (osapi:UUID)
d17e2807
M (DELETE)
d17e6444
M (GET)
d17e2589
M (GET)
d17e6370
ε
ε
U (detail)
d17e6265
ε
d17e2699
M (GET)
d17e6162
ε
U (flavors)
ε
U (osapi:UUID)
d17e3948
M (GET)
d17e8162
U (limits)
d17e3754
M (GET)
d17e7977
ε
U (detail)
ε
d17e3852
M (GET)
d17e8070
d17e248
M (GET)
d17e5940
ε

Compute API

Alternate Representation



START

v1\.1

extensions

GET
(List Extensions)

.*
(alias)

GET
(Get Extension)

GET
(Version Details)

.*
(tenantId)

images

GET
(List Images)

osapi:UUID
(imageId)

GET
(Get Image Details)

DELETE
(Delete Image)

metadata

osapi:MetadataKey
(key)

DELETE
(Delete Metadata Item)

PUT
(Create or Replace Metadata)

GET
(Get Metadata Item)

POST
(Update Metadata)

PUT
(Create or Update Metadata Item)

GET
(List Metadata)

detail

GET
(List Detail Images)

flavors

GET
(List Flavors)

detail

GET
(List Detail Flavors)

osapi:UUID
(flavorId)

GET
(Get Flavor Details)

limits

GET
(List Limits)

osapi:MetadataKey
(key)

servers

detail

GET
(List Detail Servers)

osapi:UUID
(serverId)

GET
(Get Server Details)

PUT
(Update Server Name / Administrative Password)

DELETE
(Delete Server)

metadata

action

POST
(ε)

POST
(Reboot Server)

POST
(Rebuild Server)

POST
(Resize Server)

POST
(Confirm Resized Server)

POST
(Revert Resized Server)

POST
(Create Image)

POST
(Change Password)

ips

.*
(networkId)

GET
(List Addresses by Network)

GET
(List Addresses)

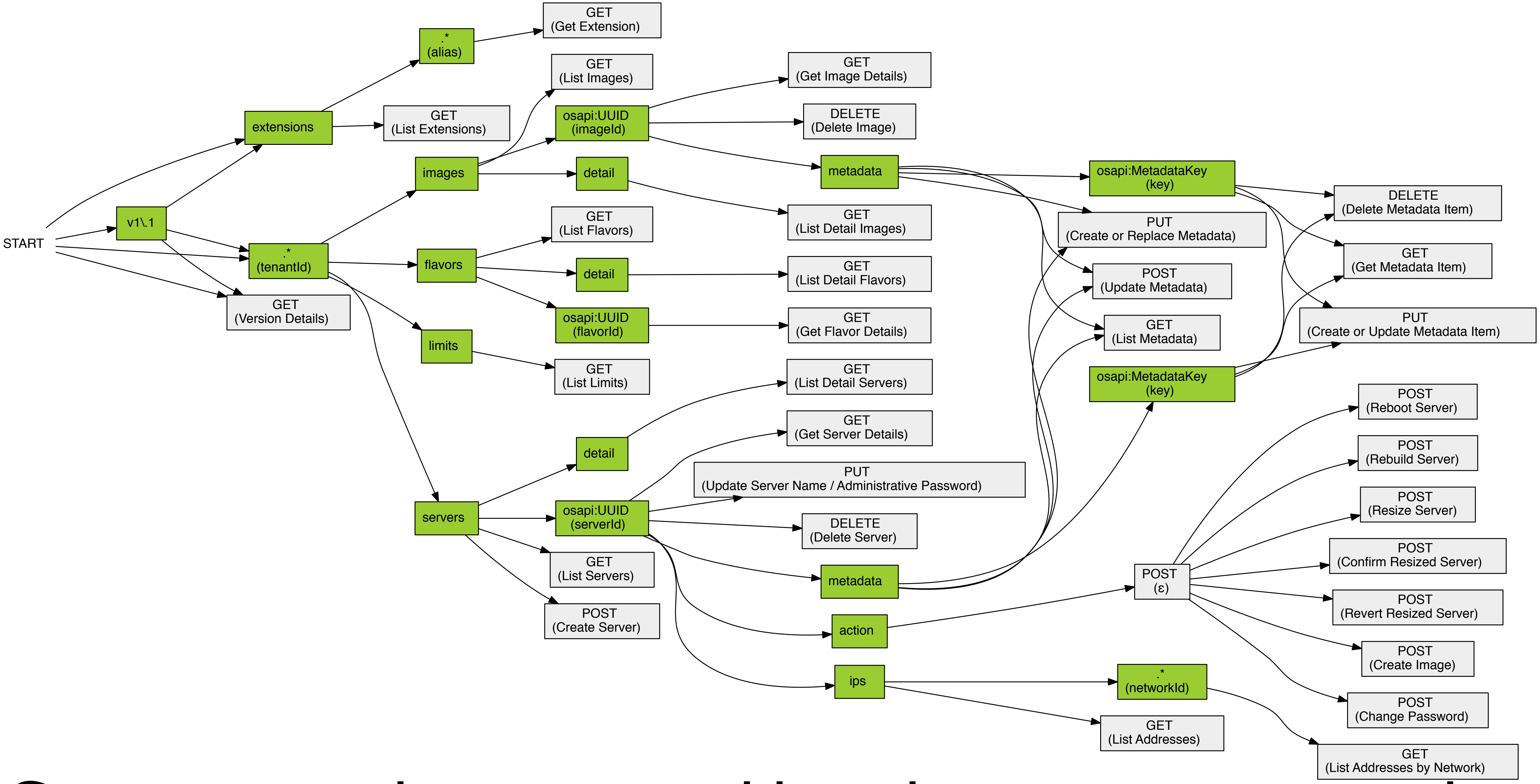
GET
(List Servers)

POST
(Create Server)

Monday, March 12, 2012

Compute API

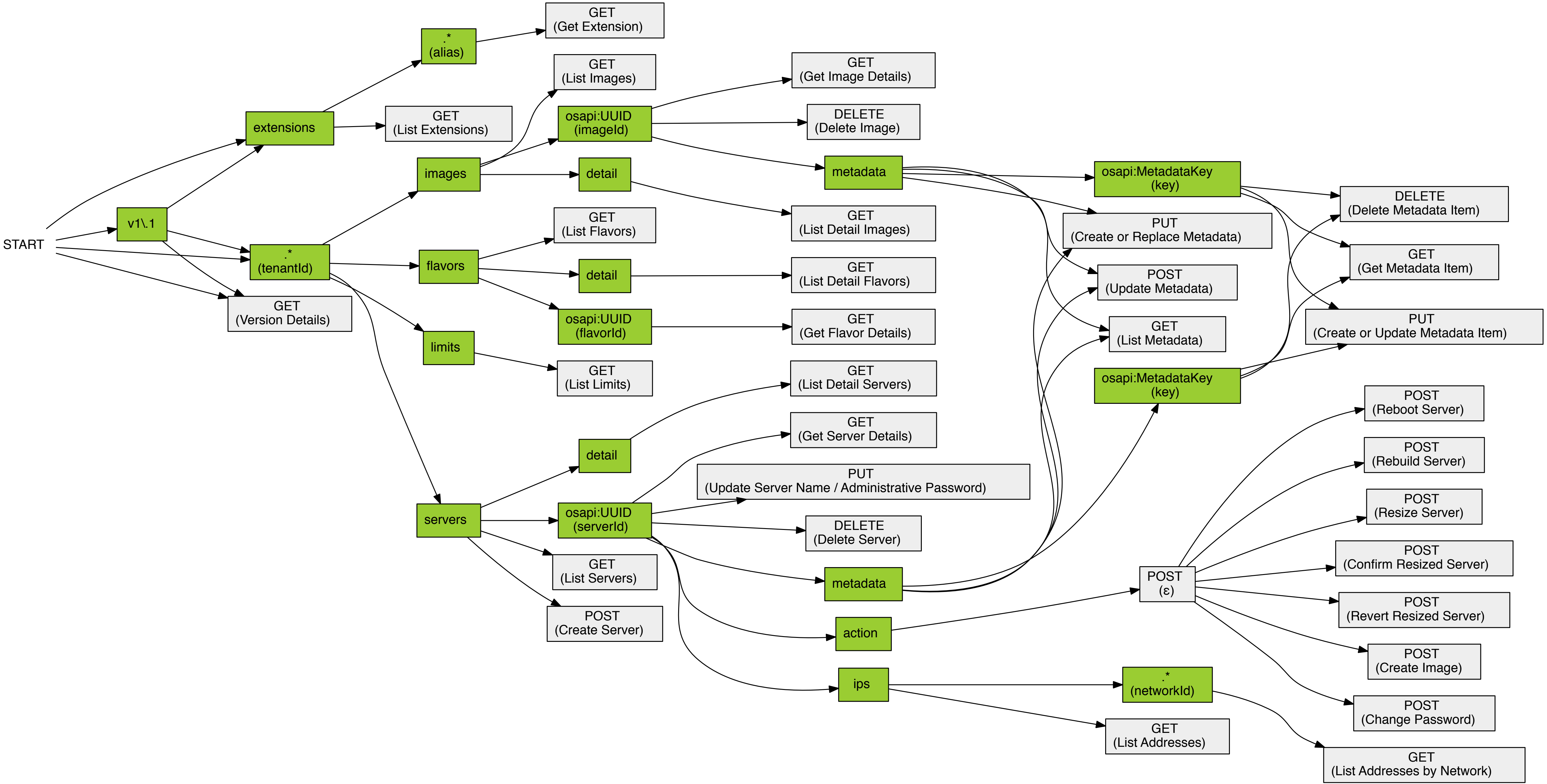
Alternate Representation



Coverage can be computed by taking into account what states have been visited...

Compute API

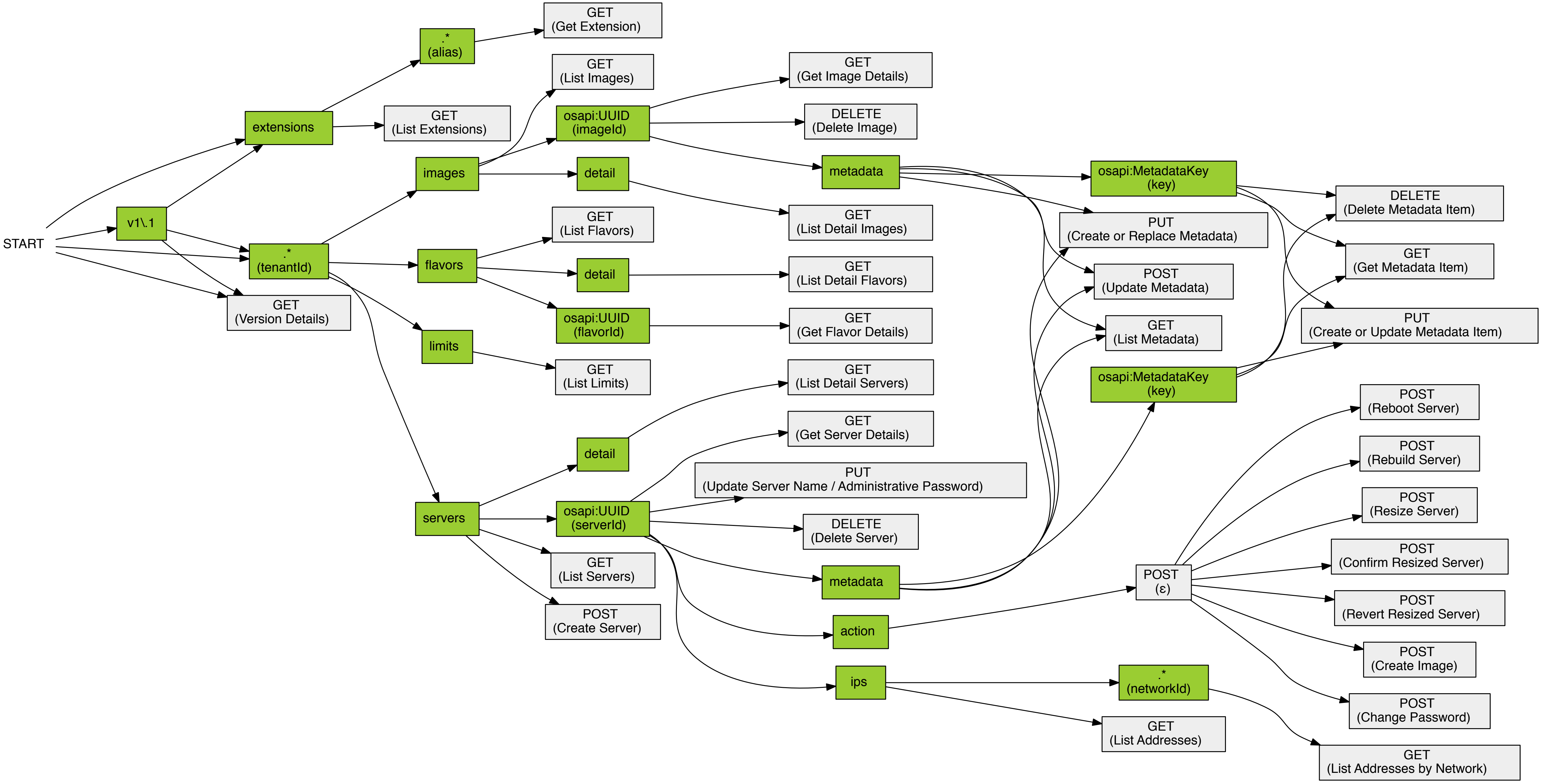
Alternate Representation



...but for the purpose of Contract Scope there are a lot of redundant states...

Compute API

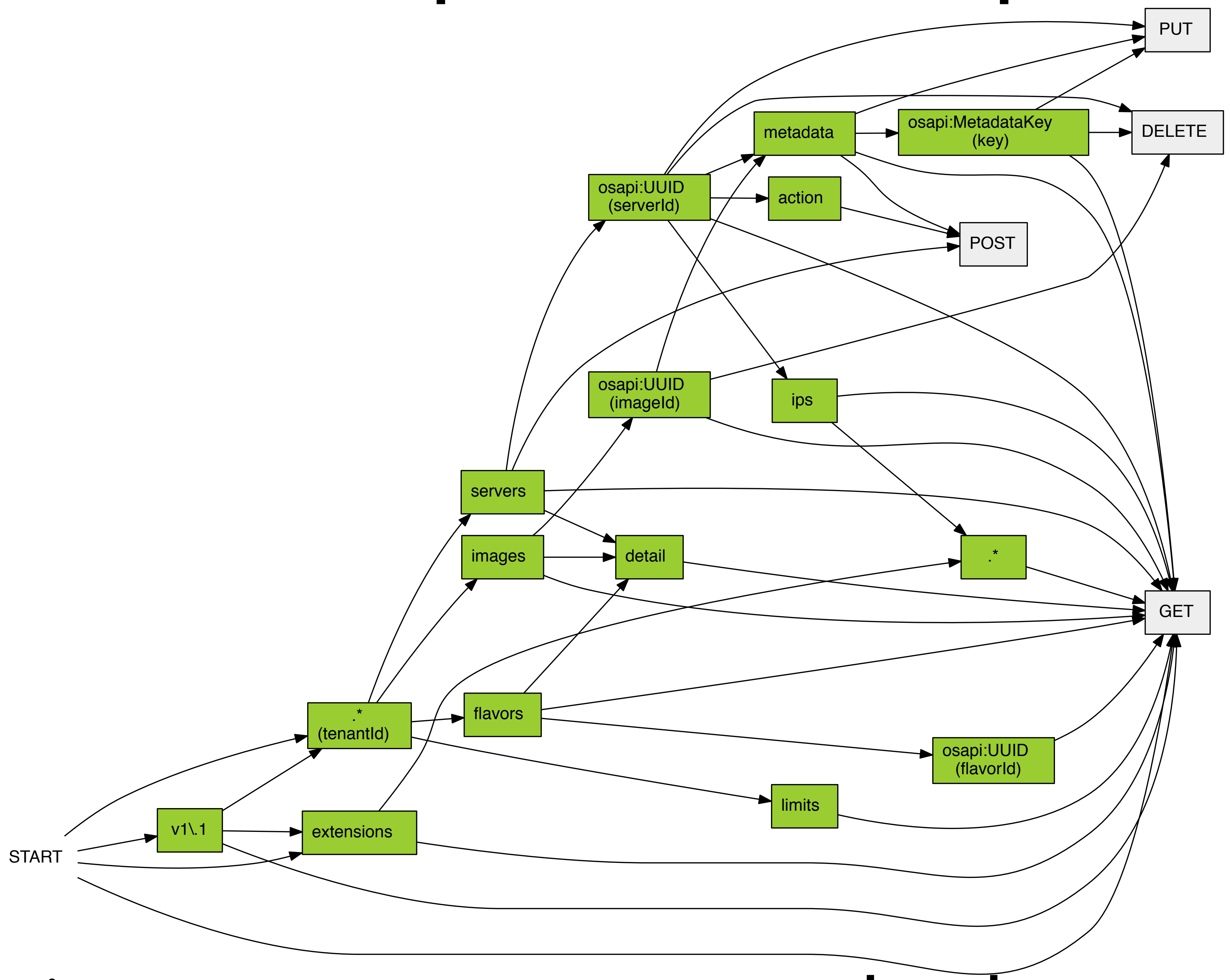
Alternate Representation



...more states mean more memory...

Compute API

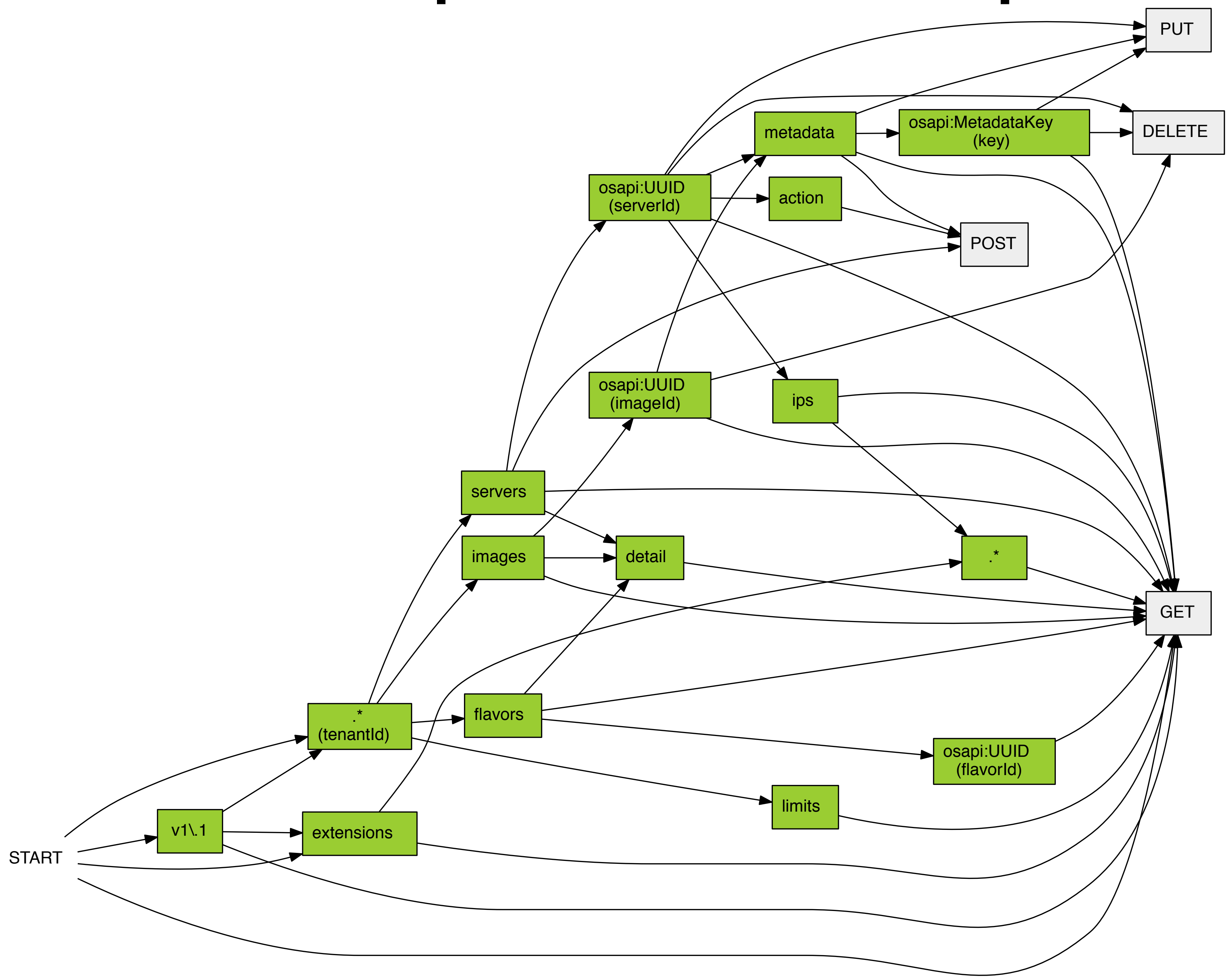
Alternate Representation, optimized



...optimization stage can remove redundant states. Great for contract scope, less states...less memory

Compute API

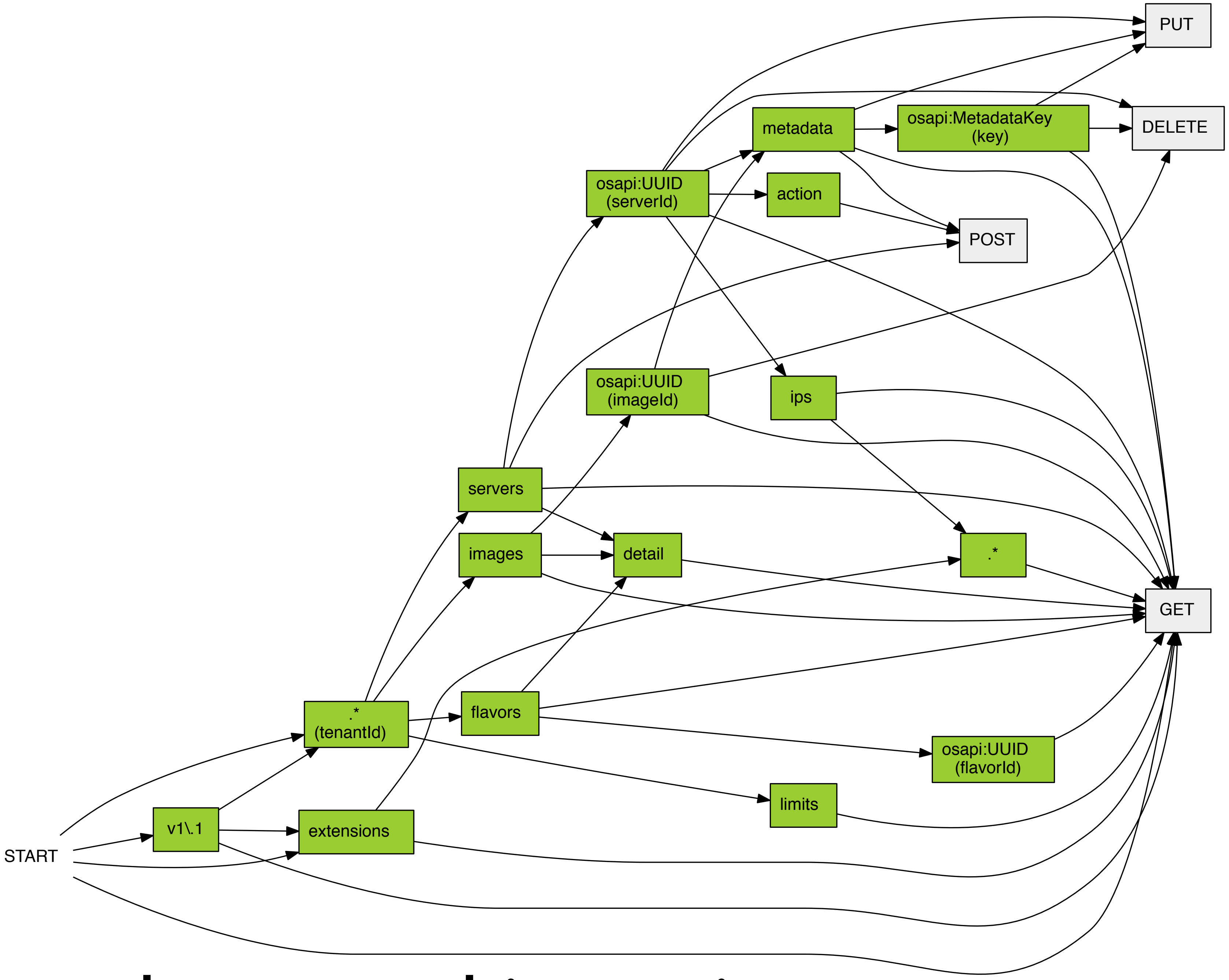
Alternate Representation, optimized



...other optimization stages are possible...

Compute API

Alternate Representation, optimized



Each stage takes a machine as input, returns a machine with less steps, shorter paths as output

What’s Available Today

- Checks on URI and Method.
- Full support for Template parameters -- taking XSD 1.1 simple types into account.
- Optimization to compress redundant nodes into a single node.
- The foundation for Contract Scope is Available today!

What's Missing for Contract Scope

- Full integration with Repose: Configuration Service Etc.
- Business Logic for Mapping Role/Group to a Validator

What’s Missing for API Coverage

- Support for Parameters, Headers, Request/ Response Content Validation

What’s Next

- Full Repose Integration
- Continue to iterate towards API Coverage Functionality
- Additional Tests/Documentation of Existing Functionality

Thanks!