A Quick Look At Spyne

Burak Arslan
burak at arskom dot com dot tr

April 7, 2013

What is Spyne?

Spyne makes it convenient to

expose your services using multiple

protocols and/or transports.

What is Spyne?

It also forces you to have a well-defined api.

How?

Here's a simple function:

```
from datetime import datetime

def get_utc_time():
    return datetime.utcnow()
```

Now, to make this function remotely callable;

Now, to make this function remotely callable;

1)

We wrap it in a `ServiceBase` subclass:

```
def get_utc_time():
    return datetime.utcnow()
```

```
from spyne.model.primitive import DateTime
from spyne.decorator import srpc
from spyne.service import ServiceBase


    def get_utc_time():
        return datetime.utcnow()
```

```
from spyne.model.primitive import DateTime
from spyne.decorator import srpc
from spyne.service import ServiceBase

class DateTimeService(ServiceBase):

    def get_utc_time():
        return datetime.utcnow()
```

```
from spyne.model.primitive import DateTime
from spyne.decorator import srpc
from spyne.service import ServiceBase

class DateTimeService(ServiceBase):
    @srpc(_returns=DateTime)
    def get_utc_time():
        return datetime.utcnow()
```

2)

Now, we have to wrap the service definition

in an `Application` definition.

```
[DateTimeService],
```

```
from spyne.application import Application
from spyne.protocol.http import HttpRpc

httprpc = Application(
          [DateTimeService],
```

```
from spyne.application import Application
from spyne.protocol.http import HttpRpc

httprpc = Application(
        [DateTimeService],
        tns='spyne.examples.multiprot',
```

```
from spyne.application import Application
from spyne.protocol.http import HttpRpc

httprpc = Application(
        [DateTimeService],
        tns='spyne.examples.multiprot',
        in_protocol=HttpRpc(),
        out_protocol=HttpRpc()
    )
```

3)

Finally, we wrap the application in

a transport.

```
from spyne.server.wsgi import WsgiApplication

application = WsgiApplication(httprpc)
```

This is now a regular WSGI Application that

we can pass to WSGI-compliant servers like

CherryPy, mod_wsgi, Twisted, etc.

```
from spyne.server.wsgi import WsgiApplication

application = WsgiApplication(httprpc)
```

This is now a regular WSGI Application that

we can pass to WSGI-compliant servers like

CherryPy, mod_wsgi, Twisted, etc.

```
$ curl http://localhost:9910/get_utc_time
2012-03-09T17:38:11.997784
```

Now, what if we wanted to expose this function using another protocol?

For example: SOAP

```
from spyne.application import Application
from spyne.protocol.http import HttpRpc
from spyne.protocol.soap import Soap11

soap = Application([DateTimeService],
        tns='spyne.examples.multiprot',
        in_protocol=HttpRpc(),
        out_protocol=Soap11()
    )
```

For example: SOAP

```
$ curl http://localhost:9910/get_utc_time \
                                    | tidy -xml -indent
```

```
<?xml version='1.0' encoding='utf-8'?>
<senv:Envelope xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing"
 xmlns:tns="spyne.examples.multiple_protocols"
 xmlns:plink="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
 xmlns:xop="http://www.w3.org/2004/08/xop/include"
 xmlns:senc="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:s12env="http://www.w3.org/2003/05/soap-envelope/"
 xmlns:s12enc="http://www.w3.org/2003/05/soap-encoding/"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:senv="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
  <senv:Body>
    <tns:get_utc_timeResponse>
      <tns:get_utc_timeResult>
        2012-03-06T17:43:30.894466
      </tns:get_utc_timeResult>
    </tns:get_utc_timeResponse>
  </senv:Body>
</senv:Envelope>
```

Or, just XML:

```
from spyne.application import Application
from spyne.protocol.http import HttpRpc
from spyne.protocol.xml import XmlDocument

xml = Application([DateTimeService],
          tns='spyne.examples.multiprot',
          in_protocol=HttpRpc(),
          out_protocol=XmlDocument()
      )
```

Or, just XML:

```
$ curl http://localhost:9910/get_utc_time \
                                    | tidy -xml -indent

<?xml version='1.0' encoding='utf-8'?>
<ns0:get_utc_timeResponse
xmlns:ns0="spyne.examples.multiple_protocols">
  <ns0:get_utc_timeResult>
    2012-03-06T17:49:08.922501
  </ns0:get_utc_timeResult>
</ns0:get_utc_timeResponse>
```

Or, HTML:

```
from spyne.application import Application
from spyne.protocol.http import HttpRpc
from spyne.protocol.xml import HtmlMicroFormat

html = Application([DateTimeService],
        tns='spyne.examples.multiprot',
        in_protocol=HttpRpc(),
        out_protocol=HtmlMicroFormat()
    )
```

Or, HTML:

```
$ curl http://localhost:9910/get_utc_time \
                                    | tidy -xml -indent

<div class="get_utc_timeResponse">
  <div class="get_utc_timeResult">
    2012-03-06T17:52:50.234246
  </div>
</div>
```

etc…

Spyne also makes it easy to implement custom protocols.

Let's implement an output protocol that renders the datetime value as an analog clock.

(without going into much detail ☺)

To do that, we need to implement the `serialize` and `create_out_string` functions in a `ProtocolBase` subclass.

```
from lxml import etree
from spyne.protocol import ProtocolBase

class SvgClock(ProtocolBase):
  mime_type = 'image/svg+xml'
```

```
from lxml import etree
from spyne.protocol import ProtocolBase

class SvgClock(ProtocolBase):
  mime_type = 'image/svg+xml'

  def serialize(self, ctx, message):
    d = ctx.out_object[0] # the return value
```

```python
from lxml import etree
from spyne.protocol import ProtocolBase

class SvgClock(ProtocolBase):
  mime_type = 'image/svg+xml'

  def serialize(self, ctx, message):
    d = ctx.out_object[0] # the return value

    # (some math and boilerplate suppressed)
```

```
from lxml import etree
from spyne.protocol import ProtocolBase

class SvgClock(ProtocolBase):
  mime_type = 'image/svg+xml'

  def serialize(self, ctx, message):
    d = ctx.out_object[0] # the return value

    # (some math and boilerplate suppressed)

    # clock is a svg file parsed as lxml Element
    ctx.out_document = clock
```

```
from lxml import etree
from spyne.protocol import ProtocolBase

class SvgClock(ProtocolBase):
  mime_type = 'image/svg+xml'

  def serialize(self, ctx, message):
    d = ctx.out_object[0] # the return value

    # (some math and boilerplate suppressed)

    # clock is a svg file parsed as lxml Element
    ctx.out_document = clock

  def create_out_string(self, ctx, charset=None):
    ctx.out_string = [
        etree.tostring(ctx.out_document)
    ]
```

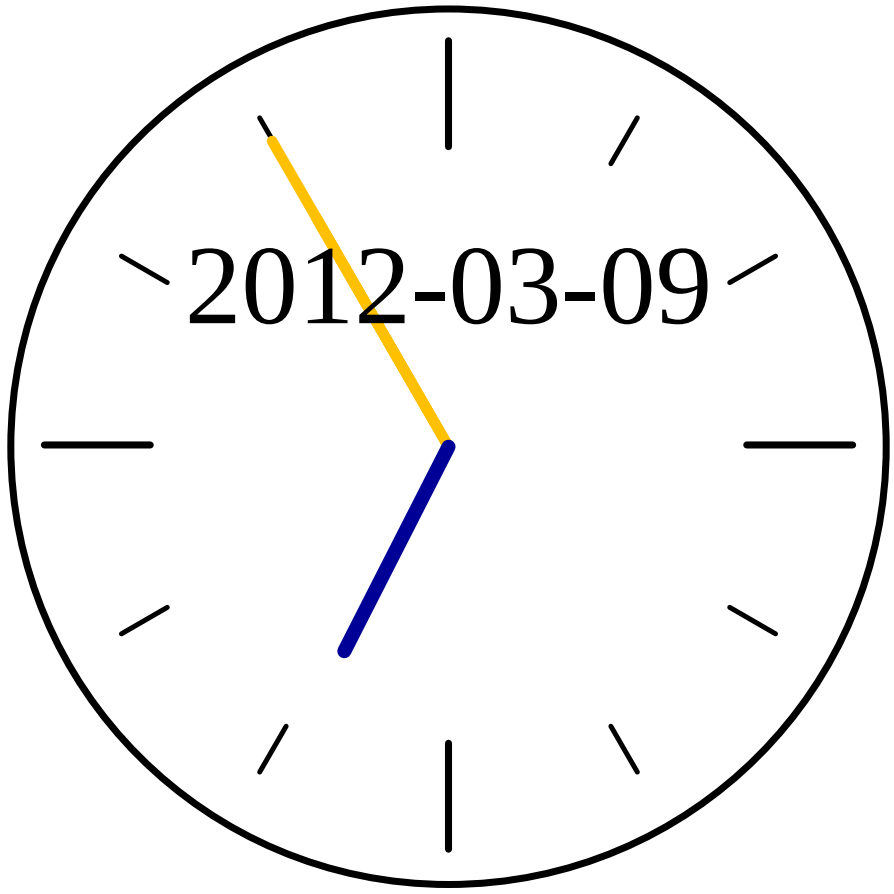
The custom SVG protocol:

```
from spyne.application import Application

svg = Application([DateTimeService],
          tns='spyne.examples.multiprot',
          in_protocol=HttpRpc(),
          out_protocol=SvgClock()
      )
```

The custom SVG protocol:

```
$ curl http://localhost:9910/get_utc_time \
                                    > utc_time.svg
```



It's also easy to implement declarative restrictions on your input data.

So instead of doing this:

```
def get_name_of_month(month):
    """Takes an integer between 1-12 and
    returns the name of month as string
    """

    value = int(month)

    if not (1 <= value <= 12):
        raise ValueError(value)

    return datetime(2000,month,1).strftime("%B")
```

You can do this:

```
class NameOfMonthService(ServiceBase):
  @srpc(Integer(ge=1,le=12), _returns=Unicode)
  def get_name_of_month(month):
    return datetime(2000,month,1).strftime("%B")
```

And if you enable validation;

```
from spyne.application import Application
from spyne.protocol.http import HttpRpc

rest = Application([NameOfMonthService],
        tns='spyne.examples.multiprot',
        in_protocol=HttpRpc(validator='soft'),
        out_protocol=HttpRpc()
    )
```

```
$ curl localhost:9912/get_name_of_month?month=3
March
```

```
$ curl localhost:9912/get_name_of_month?month=3
March
```

```
$ curl -D - localhost:9912/get_name_of_month?month=13
HTTP/1.0 400 Bad Request
Date: Sat, 10 Mar 2012 14:21:36 GMT
Server: WSGIServer/0.1 Python/2.7.2
Content-Length: 63
Content-Type: text/plain

Client.ValidationError

The string '13' could not be validated
```

So, what's missing?

Protocols: JSON! ProtoBuf! XmlRpc! Thrift!
YAML! HTML! (The whole document)

Transports: SMTP! Files! SPDY! WebSockets!

(and many other things! see the ROADMAP.rst in the source repo.)

Additional Information:

github.com/arskom/spyne

This example and the presentation are in:
examples/multiple_protocols
examples/validation.py

Stay for the sprints! I'll be around!