Introduction to Data Generator

Mohamed Ibrahim
FINRA

Data Generator ~ the basics

What is Data Generator

Data generator is a library that helps in generating patterns based on a given model.

Not a random data generator.

Can execute the model in parallel either locally or in a distributed way.

Tested to generate few Terabytes in less than a business day.

Data Generator ~ the basics

What areas are data generator useful in ?

- In developing load tests
- In developing functional tests
- In developing UI automation tests, where the generated data can represent a user scenario
- To aid in development in situations where no data exists

Data Generator ~ the basics

Data Generator is a Java library

To make best use of Data Generator, expect to write code that calls the Data Generator core library at minimum.

The core library provides tools that allow the user to:

- Load an parse a model
- Split the model into n smaller problems
- Execute each smaller problem in a separate thread
- Send the generated patterns to user defined class called ***a writer***.

Data Generator ~ the basics

How the Data Generator model looks like

Data Generator uses SCXML an XML based language that can represent complex state machines called **S**tate **C**hart e**X**tensible **M**arkup **L**anguage.

The model represents the data as states, which can set output variables to certain values. Transitions between the states can optionally contain conditions using which the user can control the data values.

Data Generator ~ the basics

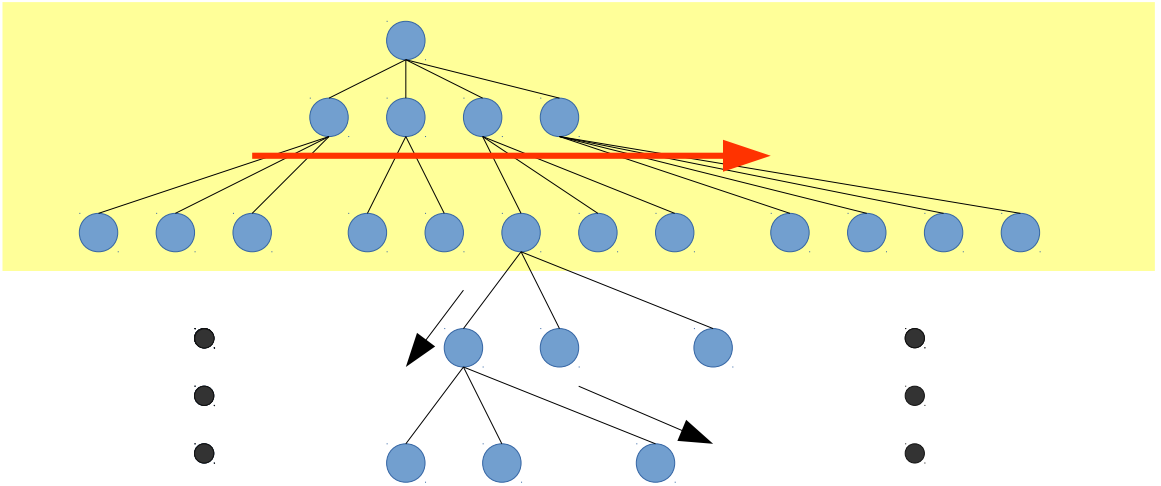
A sample SCXML model

```
01 <scxml xmlns="http://www.w3.org/2005/07/scxml"
02       xmlns:cs="http://commons.apache.org/scxml"
03       version="1.0"
04       initial="start">
05
06    <state id="start">
07        <transition event="SETV1" target="SETV1"/>
08    </state>
09
10    <state id="SETV1">
11        <onentry>
12            <assign name="var_out_V1_1" expr="set:{A1,B1,C1}"/>
13            <assign name="var_out_V1_2" expr="set:{A2,B2,C2}"/>
14            <assign name="var_out_V1_3" expr="77"/>
15        </onentry>
16        <transition event="SETV2" target="SETV2"/>
17    </state>
18
19    <state id="SETV2">
20        <onentry>
21            <assign name="var_out_V2" expr="set:{1,2,3}"/>
22            <assign name="var_out_V3" expr="#{customplaceholder}"/>
23        </onentry>
24        <transition event="end" target="end"/>
25    </state>
26
27    <state id="end">
28        <!-- We're done -->
29    </state>
30 </scxml>
```

```
01 SCXML Header
02
03
04
05
06 The start state
07
08
09 Create a state called SETV1
10   Once the execution enters this state
11     Set variables var_out_V1_1 & V1_2 to one value of the given
12
13     V1_3 is always 77
14
15
16   Unconditionally, transition to SETV2
17
18
19 Create a state called SETV2
20   Once the execution enters this state
21     Set variable var_out_V2 to one value of the given values
22     Set var_out_V3 to a template, to be replaced in the java code
23
24
25   Unconditionally, transition to END
26
27 Create a state called END
28
29
30 SCXML Closed tag
```

Data Generator ~ the basics

Engines: engines know about your model. For SCXML, we need to use an engine that knows SCXML. Engines will load and parse the model, convert it into a graph, walk the top of the graph using Depth First Search, and convert the result into a list of Frontiers that know its format – in our example an SCXMLFrontier.



Frontier: The frontiers gets serialized using a Gapper class into strings. Gappers need to know the format. In our example an SCXMLGapper is used.

Data Generator ~ the basics

Distributors: Send the String serialized Frontiers to their compute location.

Threads if you're executing locally (DefaultDistributor).

If you're using Hadoop, then they're sent to a Map job (HDFSDistributor in MR example).

Frontiers again: Once the Frontiers are reconstructed from Strings, they are asked to continue searching the graph using Depth First Search.

Once a frontier completes one round of Depth First Search, it places its findings on a Queue. The Distributor reads the Queue using a different thread and asks a list of provided transformers to operate on every item.

Transformers: Transformers convert custom place holders that you write in your model with their values. For example a value like #AccountNumber can be formatted using a transformer into 101X400.

Writers: Multiple writers provided by the user format the result and write them into multiple output files.

Data Generator ~ the basics

```
//will default to samplemachine, but you could specify a different file if you
choose to
InputStream is = CmdLine.class.getResourceAsStream("/" + (args.length == 0 ?
"samplemachine" : args[0]) + ".xml");

engine.setModelByInputFileStream(is);

// Usually, this should be more than the number of threads you intend to run
engine.setBootstrapMin(1);

//Prepare the consumer with the proper writer and transformer
DataConsumer consumer = new DataConsumer();
consumer.addDataTransformer(new SampleMachineTransformer());
consumer.addDataWriter(new DefaultWriter(System.out,
        new String[]{"var_out_V1_1", "var_out_V1_2", "var_out_V1_3",
"var_out_V2", "var_out_V3"}));

//Prepare the distributor
DefaultDistributor defaultDistributor = new DefaultDistributor();
defaultDistributor.setThreadCount(1);
defaultDistributor.setDataConsumer(consumer);
Logger.getLogger("org.apache").setLevel(Level.WARN);

engine.process(defaultDistributor);
```