Trace Log to Sequence Diagram Conversion

Browsing through the code traces is an exercise in frustration. Applications produce huge logs that take hours to analyze. The Python scripts presented here let you visualize the traces as sequence diagrams. Interactions between objects are presented visually.

Step 1: Sample Trace to Sequence Diagram Conversion

Let's get started by converting a sample trace into a sequence diagram. We start with downloading a few tools:

1. Download and install a free 45 day trial of EventStudio System Designer from http://www.EventHelix.com/EventStudio/.
2. Download and install Python 2.7 from http://www.python.org/getit/releases/2.7/
3. Download and extract the Trace to sequence diagram Python scripts from https://github.com/eventhelix/trace-to-sequence-diagram
4. Before we proceed, please confirm the EventStudio path setting in the config.py file:
    a. On machines running 64 bit Windows, the eventStudioPath Python variable should be set as:

```
eventStudioPath=r'"C:\Program Files (x86)\EventHelix.com\EventStudio System Designer 5\evstudio.exe"
```

    b. On 32 bits Windows platforms, eventStudioPath should be set as:

```
eventStudioPath=r'"C:\Program Files\EventHelix.com\EventStudio System Designer 5\evstudio.exe"'
```

Note: If you get "'C:\Program' is not recognized as an internal or external command, operable program or batch file" error, check if the EventStudio path is correctly specified.

5. Type cmd in the Run menu to invoke the Windows command prompt.
6. Navigate to the directory where to downloaded the Python scripts for this project.
7. On the command line now type:

    trace2sequence.py -i sample_trace.txt

8. Click on the following generated diagrams:
    a. **sequence-diagram.pdf** - A sequence diagram showing object level interactions
    b. **component-level-sequence-diagram.pdf** - A high level sequence diagram that shows high level interactions
    c. **context-diagram.pdf** - A context diagram of the object interactions.
    d. **xml-export.xml** - XML representation of the object interactions. Use this XML output to develop your custom tools.

Step 2: Customize Regular Expressions to Map Traces to FDL (config.py)

By now you would have seen a sequence diagram generated from the sample trace output. If you can modify your traces to match the trace output in sample_trace.txt you can skip this step.

In most cases however, there will be changes needed in config.py to map your trace format to the FDL input needed by EventStudio.

The config.py file lets you configure the trace to sequence diagram (FDL) mapping for:

- Remarks that are shown on the right side of a sequence diagram
- Message interactions
- Method invoke and return
- Object creation and deletion
- Action taken by an object
- State transition
- Timer start, stop and expiry
- Resource allocation and freeing

We will be mapping traces to FDL statements to generate the sequence diagrams. The mapping will also require a basic understanding of regular expressions. So let's visit these topics before we go any further.

Learning FDL - The Sequence Diagram Markup Language

FDL (Feature Description Language) will be used to generate sequence diagrams. For a quick overview of FDL, refer to the FDL sequence diagram tutorial http://www.eventhelix.com/EventStudio/sequence-diagram-tutorial.pdf.

Python Regular Expressions

The Python website has a good [introduction to regular expressions - http://docs.python.org/library/re.html.

PythonRegEx.com (http://www.pythonregex.com/) is great for testing your regular expressions.

Templates and Regular Expressions

If you browse config.py, you will see that the trace extraction regular expressions and the FDL statement generation templates are defined to next to each other. You would rarely need to change the FDL generation templates but they give you a context for defining the regular expressions.

Refer to the example below. invokeRegex regular expression extracts named fields, called, method and params. These fields along with the caller are used in the FDL template.

```
# Regular expression for parsing the function/method entry trace body
invokeRegex = '(?P<called>\w+)(\.|::)(?P<method>\w+)\s*(\((?P<params>\w+)\))?'

# FDL mapping template for function/method entry
invokeTemplate = '{caller} invokes {called}.{method}{params}'
```

Step 3: High Level Object Grouping and Bookmarks (customize.py)

Generated sequence diagrams can get complex in two ways:

1. Diagrams are long and extend into several pages
2. Too many interactive objects are present on a sequence diagram

Bookmarks

The first problem can be addressed by using bookmarks. Once you define bookmarks you will be able to navigate quickly to different parts of the sequence diagram. So add your important messages into the bookmark list. An example is shown below:

```
# Add messages that need to be bookmarked in the PDF file. This is useful
# as it lets to quickly navigate through the sequence diagram output of
# a trace. PDF quick navigation bookmarks will be added whenever the messages
# listed below are seen in the trace message.
bookmarks = frozenset({
      'RandomAccessMessage',
      'RRCConnectionSetupComplete',
      'InitialUEMessage',
      'ReleaseConnection'
      })
```

Specify Object Groupings

The second issue highlighted above is addressed by grouping objects. You specify the parents for objects in the customize.py file.

EventStudio uses this information to generate a high level sequence diagram (component-level-sequence-diagram.pdf).

An excerpt from customize.py illustrates this:

```
# EventStudio can generate a high level sequence diagram that can abstract
# out a set of classes as a high level entity. This abstraction is useful in
# understanding the trace output at a higher level of abstraction.
#
# List the interacting entities along with their parent. For example, the
# tuples below indicate that DSP_01 and DSP_23 belong to the same high level PHY
# entity. This means EventStudio will generate trace output at two levels:
# - A sequence diagram where DSP_01 and DSP_23 show up as separate axis.
# - A high level sequence diagram where PHY axis abstracts out the interactions
#   involving DSP_01 and DSP_23
objectParents = OrderedDict([
                  # Tuples of object and its parent
                  # (entity, parent)
                   ('Mobile','UE'),
                   ('DSP_01','PHY'),
                   ('DSP_23','PHY'),
                   ('RLC', 'BSC'),
                   ('MessageRouter', 'BSC'),
                   ('MobileManager', 'BSC'),
```

```
                ('CoreNetwork', 'EPC'),
                ('default-component', 'Component')
            ])
```